**GETTING STARTED WITH KEIL IDE**

**DATE:12/09/14**

**EXP NO.1**

**AIM:**

The main aim of the experiment is to understand all details of Keil µVision4 IDE. A brief overview of Keil is given as to how to create a new project, test and run the program in Keil debugger. The various windows in Keil µVision4 IDE are studied in depth and are illustrated using sample programs.

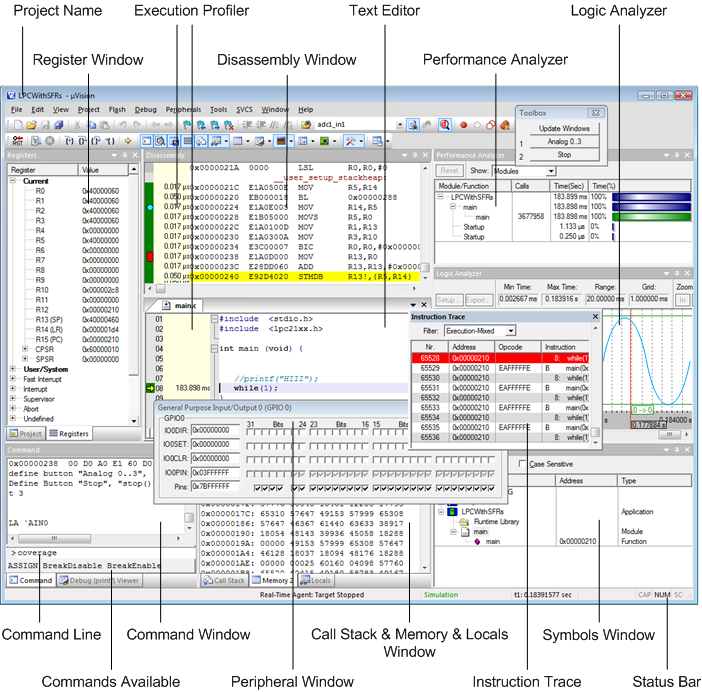
**Introduction to IDE:**

An **Integrated / Interactive Development Environment (IDE)** is a [software application](http://en.wikipedia.org/wiki/Software_application) that provides comprehensive facilities to computer programmers for [software development](http://en.wikipedia.org/wiki/Software_development). An IDE normally consists of a [source code editor](http://en.wikipedia.org/wiki/Source_code_editor), [build automation](http://en.wikipedia.org/wiki/Build_automation) tools and a [debugger](http://en.wikipedia.org/wiki/Debugger). Most modern IDEs offer [intelligent code completion](http://en.wikipedia.org/wiki/Intelligent_code_completion) features.

Some IDEs contain a [compiler](http://en.wikipedia.org/wiki/Compiler), [interpreter](http://en.wikipedia.org/wiki/Interpreter_(computing)), or both, such as [Net Beans](http://en.wikipedia.org/wiki/Net_Beans) and [Eclipse](http://en.wikipedia.org/wiki/Eclipse_(software)) others do not, such as [Sharp Develop](http://en.wikipedia.org/wiki/SharpDevelop) and [Lazarus](http://en.wikipedia.org/wiki/Lazarus_(IDE)). The boundary between an integrated development environment and other parts of the broader software development environment is not well-defined. Sometimes a [version control system](http://en.wikipedia.org/wiki/Version_control_system) and various tools are integrated to simplify the construction of a [GUI](http://en.wikipedia.org/wiki/GUI). Many modern IDEs also have a [class browser](http://en.wikipedia.org/wiki/Class_browser), an [object browser](http://en.wikipedia.org/wiki/Object_browser), and a [class hierarchy](http://en.wikipedia.org/wiki/Class_hierarchy) [diagram](http://en.wikipedia.org/wiki/Diagram), for use in [object-oriented software development](http://en.wikipedia.org/wiki/Object-oriented_programming).

Integrated development environments are designed to maximize programmer productivity by providing tight-knit components with similar [user interfaces](http://en.wikipedia.org/wiki/User_interface). IDEs present a single program in which all development is done. This program typically provides many features for authoring, modifying, compiling, deploying and debugging software. This contrasts with software development using unrelated tools, such as [vi](http://en.wikipedia.org/wiki/Vi), [GCC](http://en.wikipedia.org/wiki/GNU_Compiler_Collection) or [make](http://en.wikipedia.org/wiki/Make_(software)).

**Introduction to Keil IDE**:



***Figure 1: Keil IDE with all windows***

The µVision IDE from Keil combines project management, make facilities, source code editing, program debugging, and complete simulation in one powerful environment. The µVision development platform is easy-to-use and helping you quickly create embedded programs that work. The µVision editor and debugger are integrated in a single application that provides a seamless embedded project development environment. The µVision4 Getting Stared Guide helps you quickly create and test embedded applications for ARM, Cortex-M, C166, C251, and C51 microcontrollers.

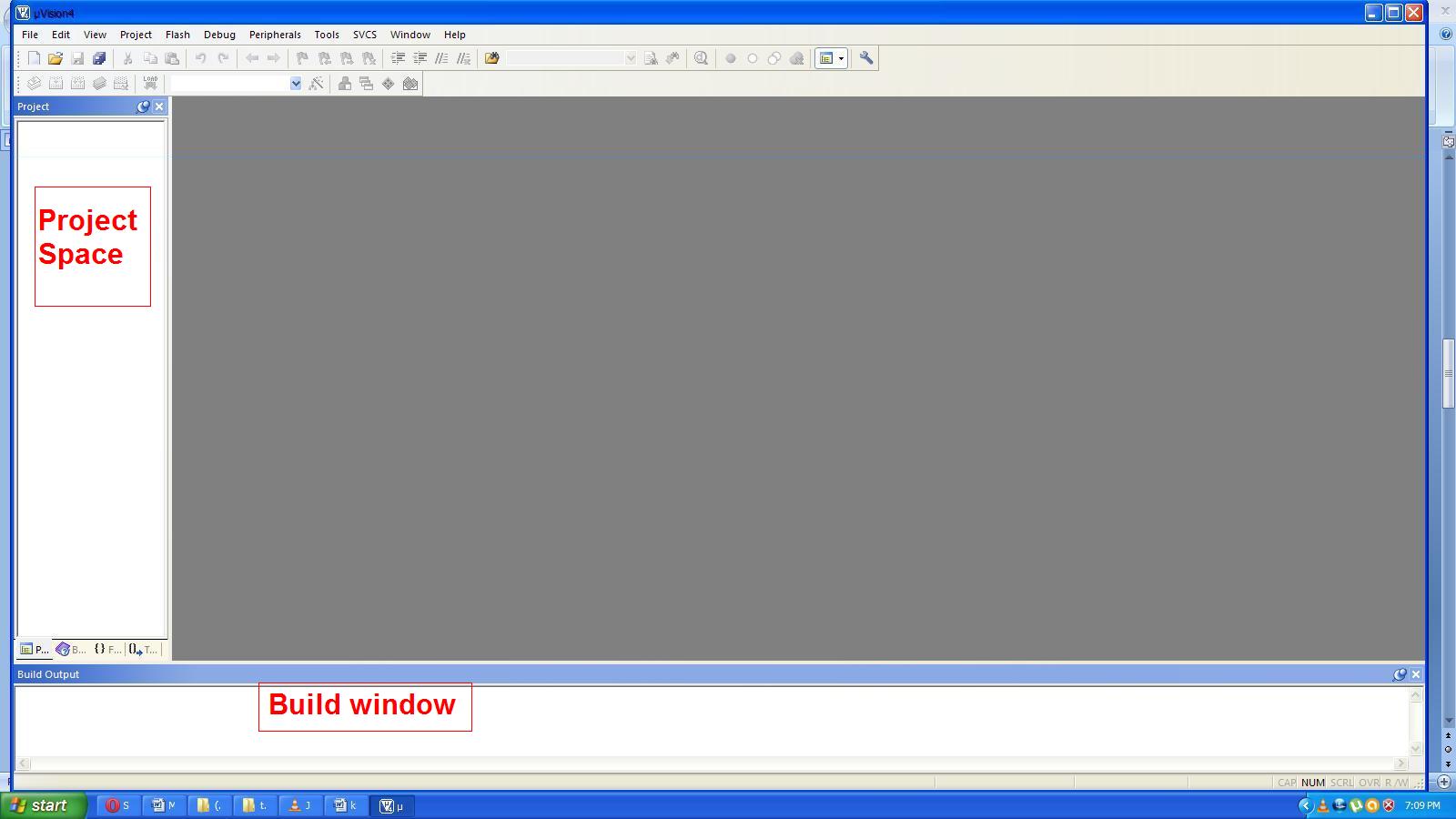
Keil µVision IDE is a freeware which supports programs with maximum size of 2kB. Though various IDEs mentioned above have their own pros and cons, Keil is a very effective tool to create embedded applications. The µVision GUI provides a menu for selecting commands and a toolbar with command buttons. Windows can be [docked](ms-its:C:\Keil\UV4\UV4.chm::/uv4_ui_docking_windows.htm) even to another physical screen. The window layout is saved for each project automatically and restored the next time the project is used.

µVision has two operating modes, the [Build Mode](ms-its:C:\Keil\UV4\UV4.chm::/uv4_creating_apps.htm) for creating applications and the [Debug Mode](ms-its:C:\Keil\UV4\UV4.chm::/uv4_debugging.htm) for analyzing applications. In addition to the menus, commands, and windows described in this chapter, the Debug Mode offers these [Windows and Dialogs](ms-its:C:\Keil\UV4\UV4.chm::/uv4_db_dbg_win_dialogs.htm).

In window layer concept, developers can define their working environment to fit the needs and preferences. For a better understanding of further comments and instructions, three major screen areas are defined. The Project Windows area is that part of the screen in which, by default, the Project Window, Functions Window, Books Window, and Registers Window are displayed. Within the Editor Windows area, you are able to change the source code, view performance and analysis information, and check the disassembly code. The Output Windows area provides information related to debugging messages, memory, symbols, call stack, local variables, commands, browse information, and find in files results. Windows can be docked to another window, to the Multiple Document Interface (MDI), or even floated to another screen. As soon as a window is dragged, several docking symbols are displayed as shown in the screenshot.

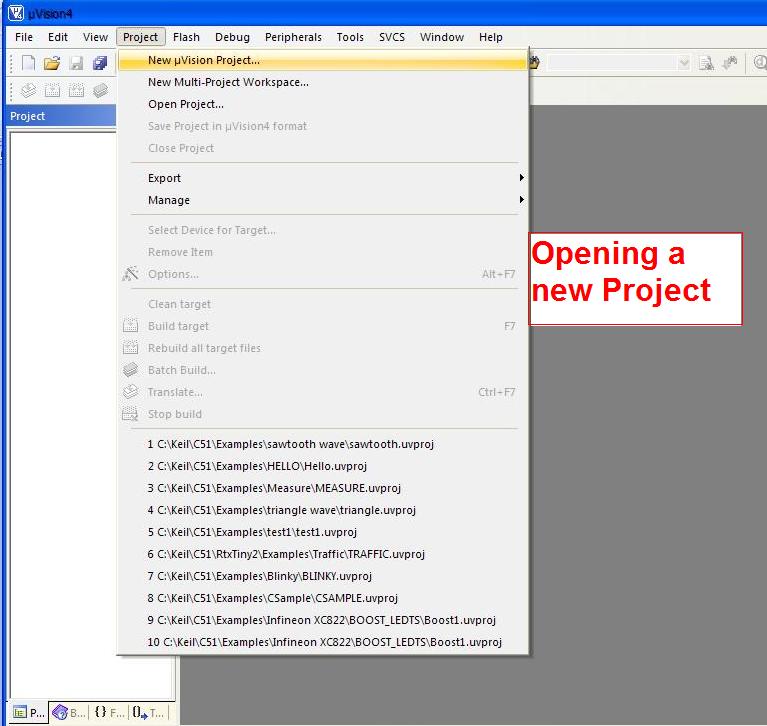
**Getting Started with Keil**:

Open the Keil µVision IDE from the start menu and it opens up as shown in figure 1.

****

***Figure 2: Keil IDE starting window***

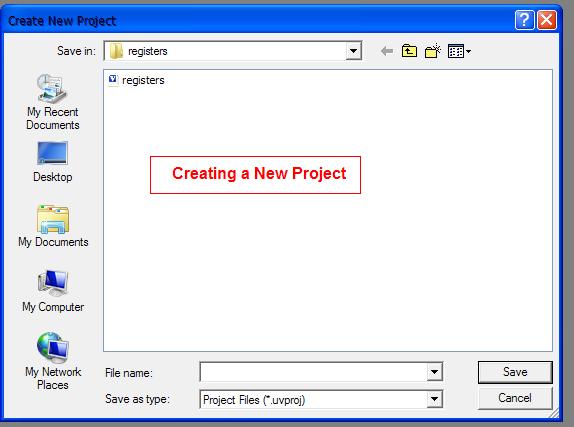
Starting a New µvision project from the Project menu.



***Figure 3: Opening a New project***

Navigate to the folder where the new project will reside. It is good practice to use a separate folder for each project.

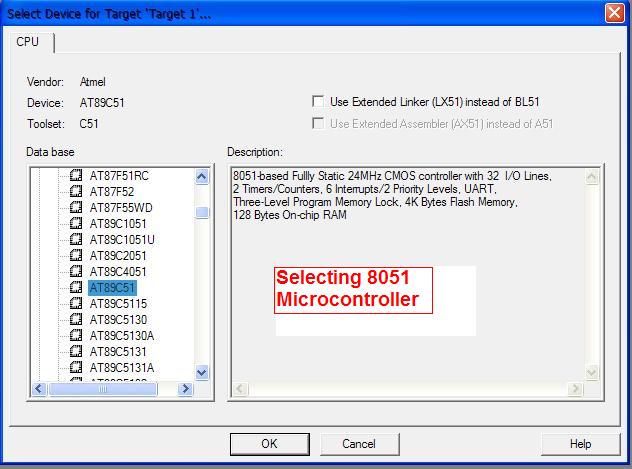
Enter the file name for the new project, for example, *Project1*. The file gets the extension *\*.UVPROJ*. The project is created with a default target- and file group name visible in the window **Project**.



***Figure 4: Creating a new Project and saving it***

### Select the Target Microcontroller from the Device Database

µVision asks to select a device for the application. Find and select the microcontroller in the dialog **Select Device for Target**. This selection sets the default device and tool configuration characteristics. For example, in this picture the ATMEL 89C51 is used. Some devices require additional [Device Database Parameters](ms-its:C:\Keil\UV4\UV4.chm::/uv4_c_dd_parameters.htm) that have to be entered manually. Please read carefully the information provided in the field Description. It might contain instructions for configuring the device.

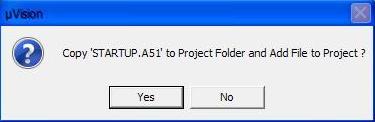


***Figure 5: Selecting a Device for Target***

### Copy and Add the Startup Code

Embedded programs require some kind of microcontroller initialization code that has to match the configuration of the hardware design. This startup code depends also on the tool chain in use. The startup code delivered with µVision configures the microcontroller device and initializes the compiler run-time system.

For most devices, µVision asks to copy the chip-specific startup code to the project. This is required for almost all projects (exceptions are: library projects and add-on projects). The file containing the startup code should be copied to the project folder, because adjustments to match the target hardware might be required. Therefore, answer this question with YES.

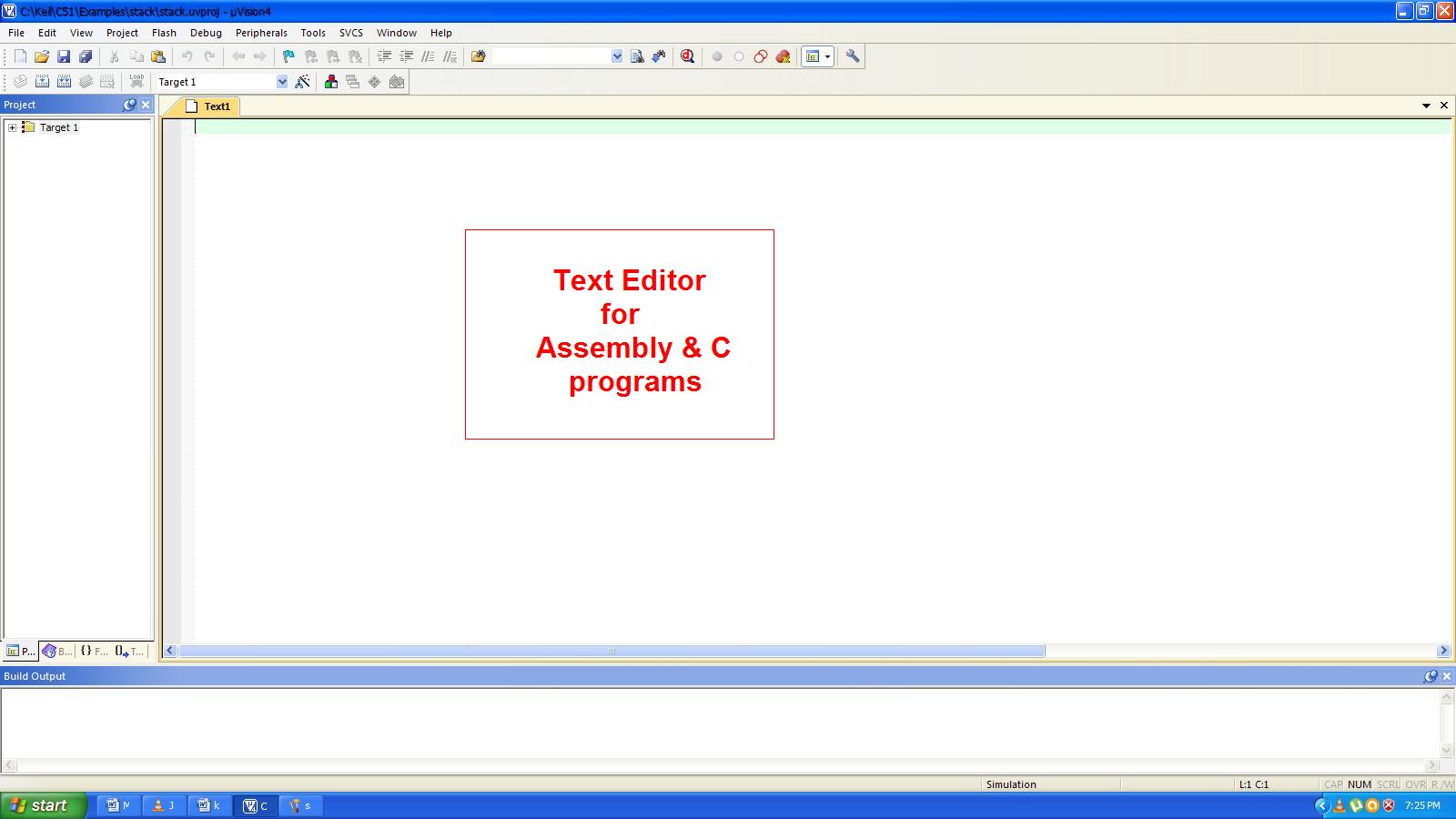


***Figure 6: Including Startup file***

**Create New Source Files**

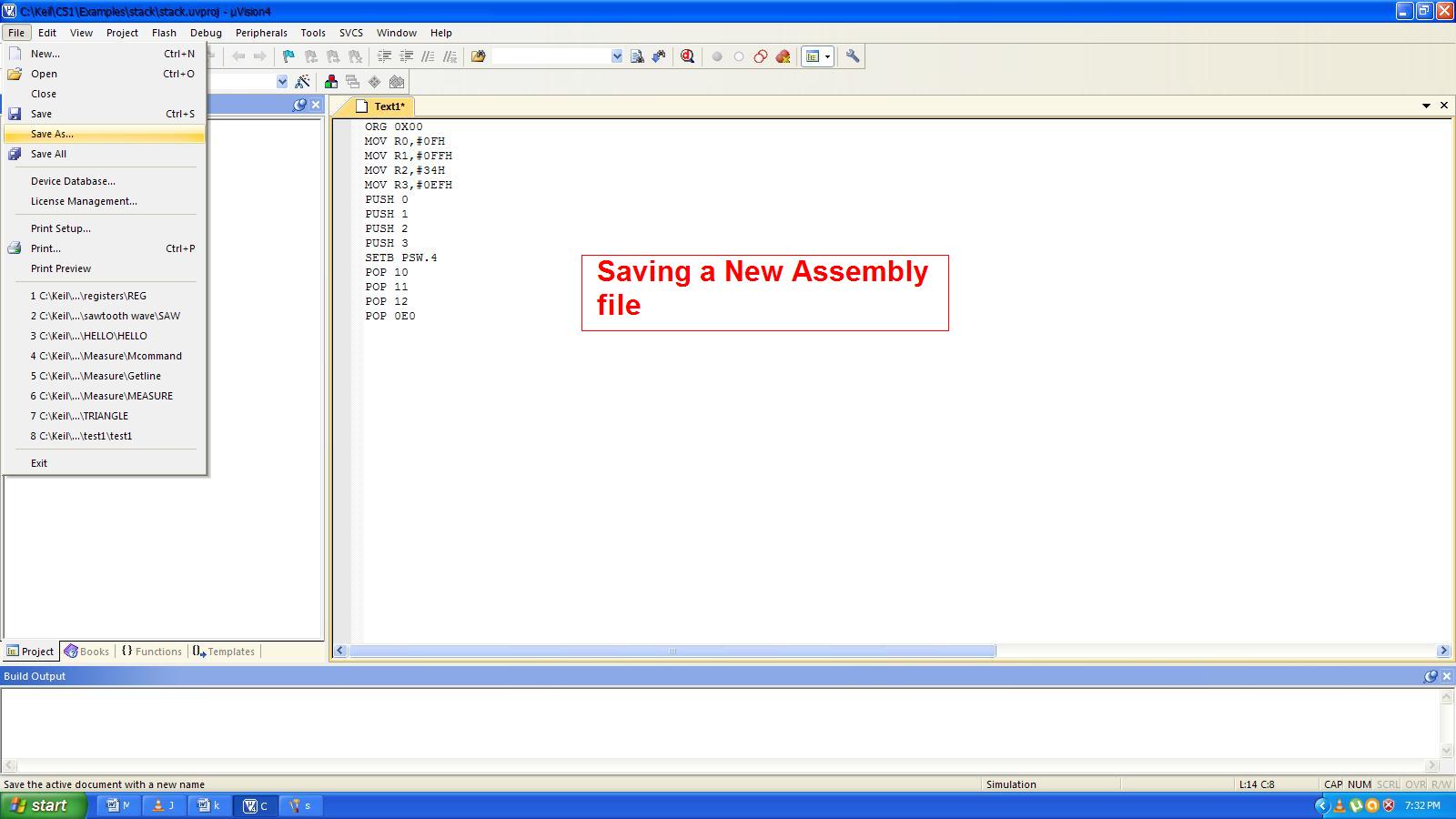
Create a new source file with the menu **File - New**. This opens an empty file in the editor window. Save the file (menu **File - Save As**) to make use of the syntax coloring feature. Syntax coloring is controlled by:

* The option Use syntax coloring in the dialog Edit - Configuration - Editor.
* The definitions in the dialog Edit - Configuration - Colors & Fonts.



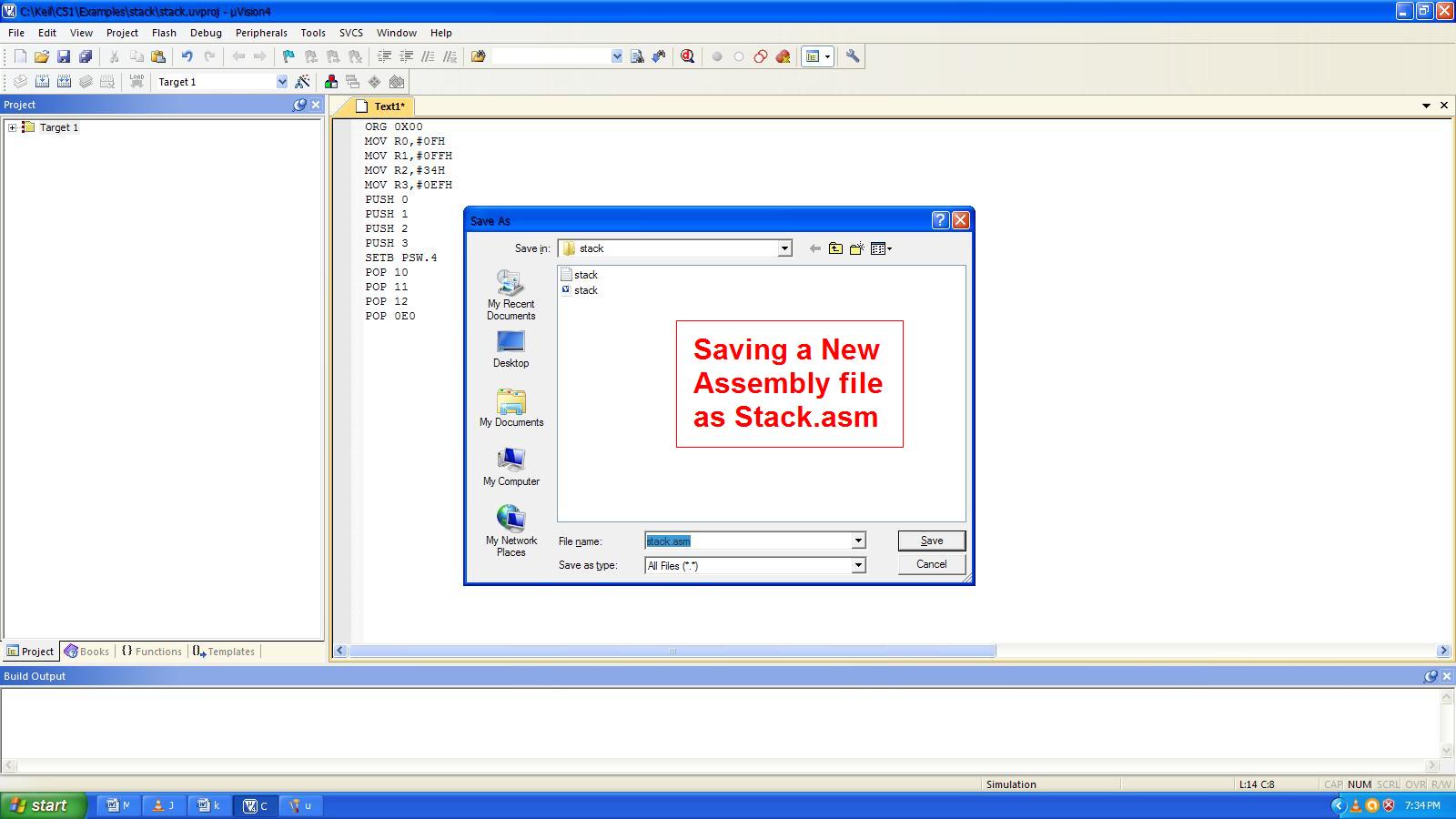
***Figure 7: Opening a new Text Editor***

In that Text editor writing the new assembly code and it will be looking as shown in the below figure 7.



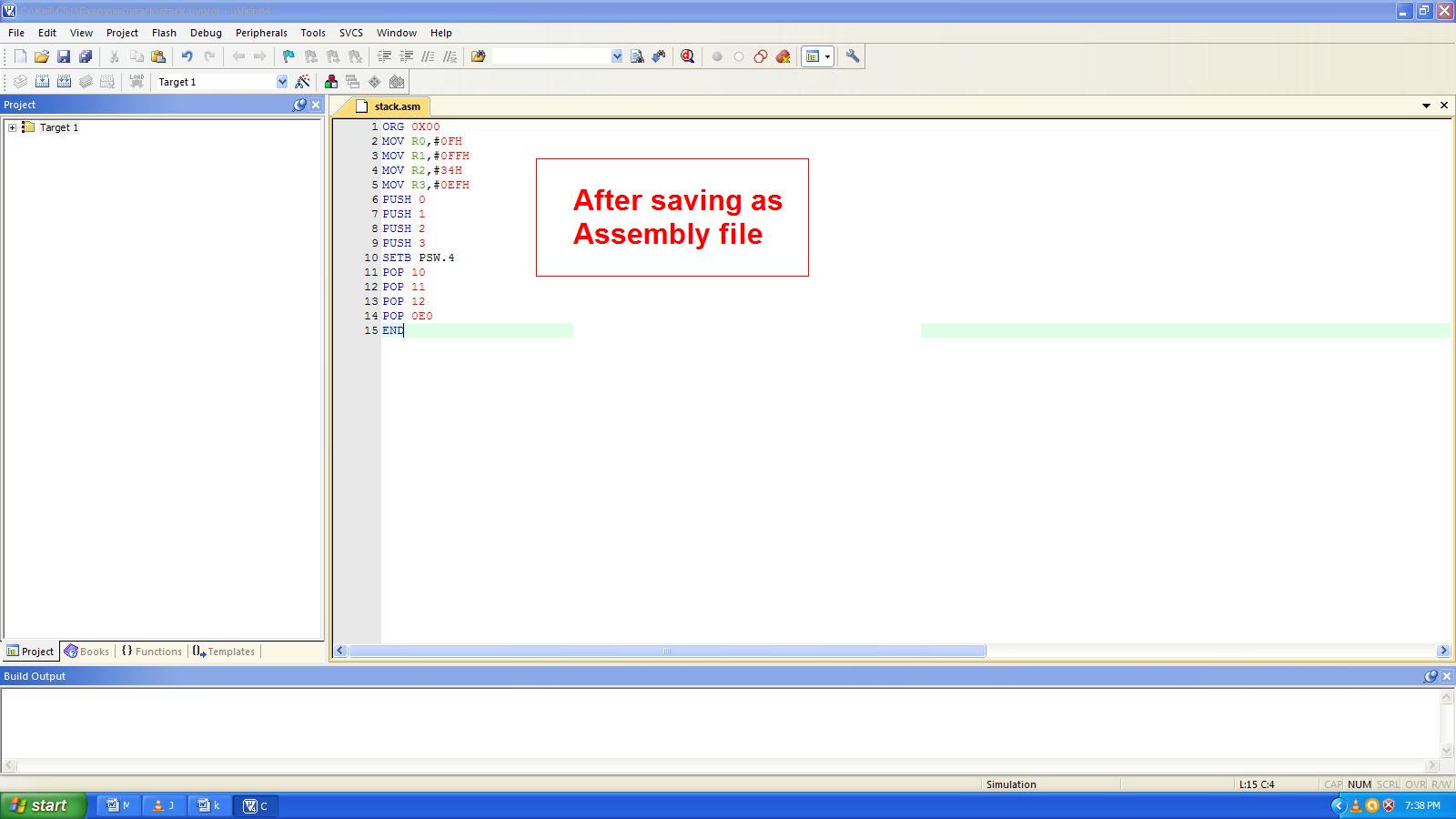
***Figure 8: Writing a new Assembly file***

After writing the assembly code save it as a .asm file.



***Figure 9: Saving a new text as .asm file***

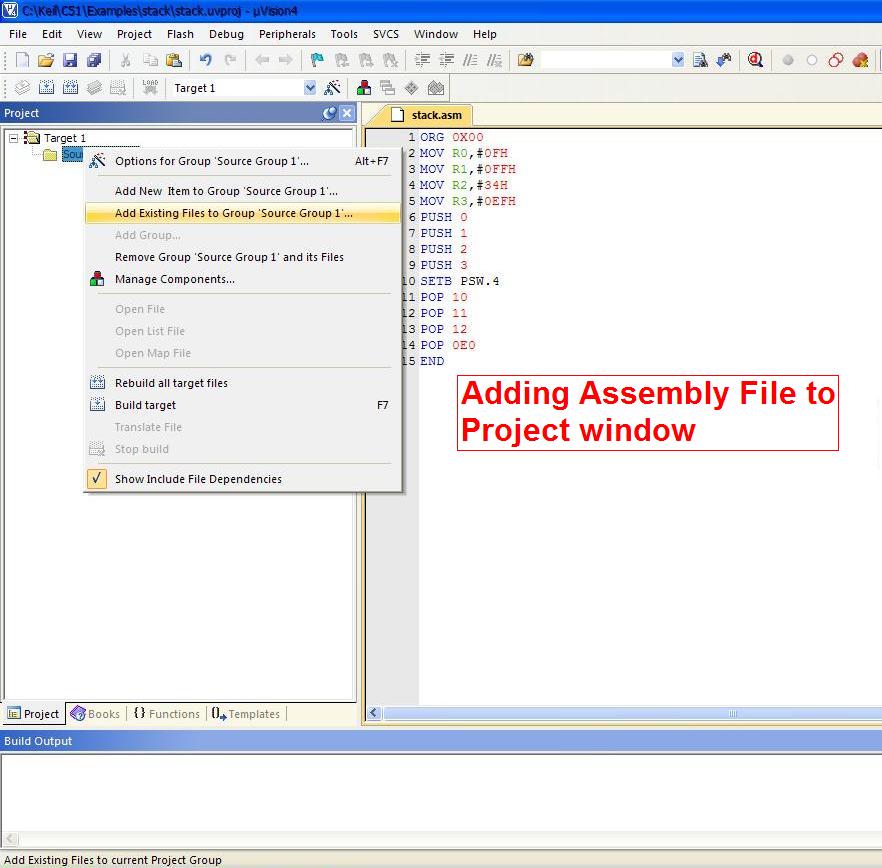
After saving it as .asm file the colors of the assembly code will be changed.



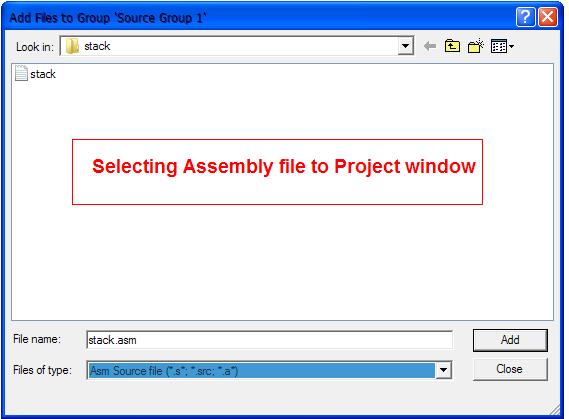
***Figure 10:After saving a Assembly File***

**Add Source Files to Project**

Several ways exist to add source files to a project. For example, click the file group in the window **Project**, open the Context Menu, and select **Add Files to Group**. This opens the standard files dialog. Select the files and click **Add**. As an alternative, use the menu Project - Manage - Components, Environment, Books - ProjectComponents, highlight the group, and click **Add Files**

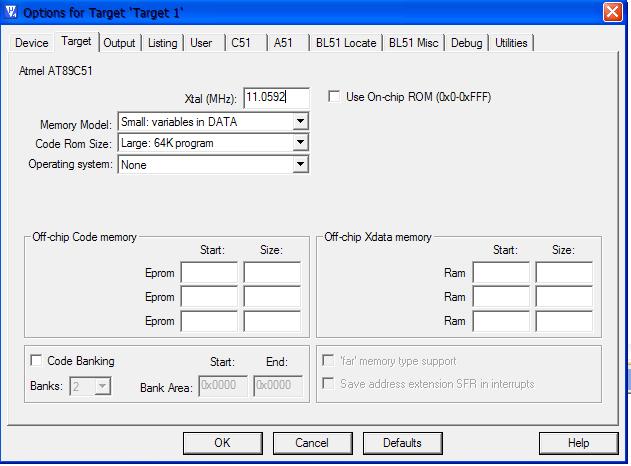


***Figure 11: Adding source file to Project window***

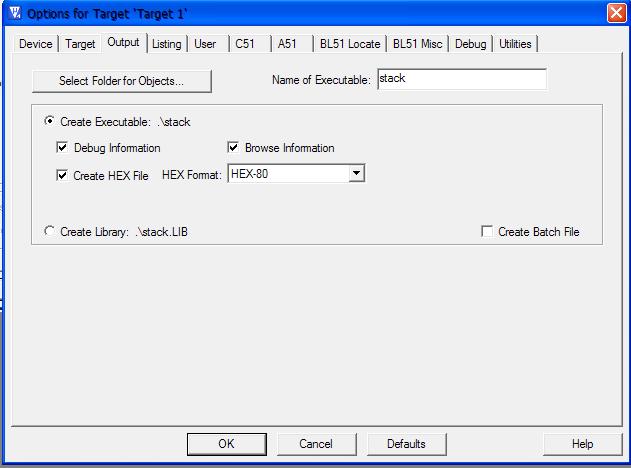


***Figure 12: Selecting a source file to Project window***

After adding source file we can check for target options and generate Hex file by building targets. Tool Options allow to configure the target hardware and the development environment. The options depend on the device and the toolchain selected for the application. Details are described in the chapter [Options](ms-its:C:\Keil\UV4\UV4.chm::/uv4_dg_options.htm). Open the dialog from the toolbar or using the menu Project — Options for Target. For example, the tab Target allows specifying all relevant parameters of the target hardware and the device's on-chip components.

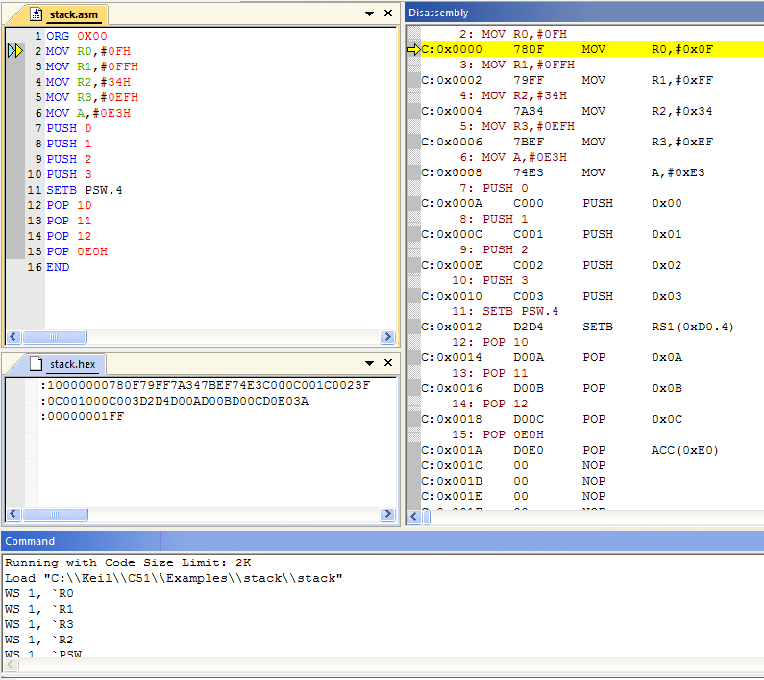


***Figure 13: Target options for Hardware utilities***



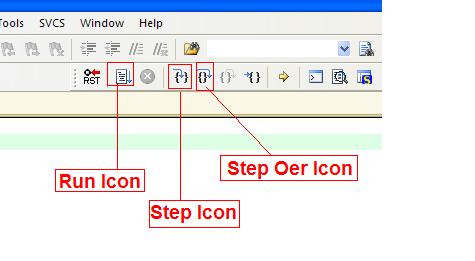
***Figure 14: In target options creating Hex file***

The file in which the program has been typed is translated and built by clicking the Translate, Build and Rebuild icons respectively. Building process compiles and assembles the source files in the project and links them together into an absolute, executable program and thus the Hex file is created and it will look like in the below figure and which can uploaded to the Microcontroller kit.

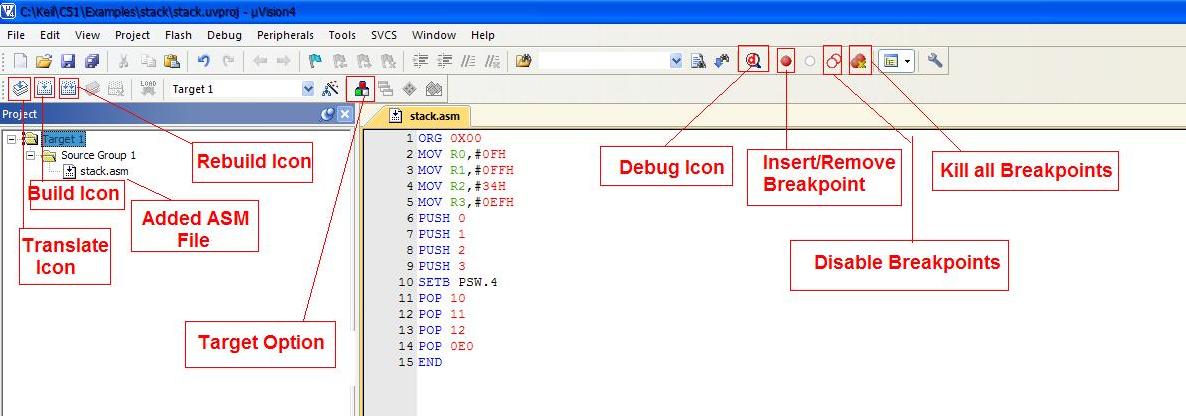
***Figure 15: HEX file***

**Various Icons in the Keil µvision IDE:**

* **Build target** **Icon** (Build Target Button ) is used to translates modified files and builds the application.
* **Rebuild all targets Icon** (Rebuild Target Button ) is used to Re-translate all source files and builds the application.
* **Translate Icon**(Translate Button ) is used to Translates the active file
* **Run Icon ( Run Button )** is used to continues executing the program until the next active breakpoint is reached.
* **Step Icon** **(Single-Step Button )** is used to executes a single-step into a function; Executes the current instruction line.
* **Step over Icon (Step Over Button )** is used toexecutes a single-step over a function.
* **Start/Stop Debug session Icon** (Start/Stop Debug Button ) is used to Starts or stops a debugging session.
* **Insert/Remove Break point Icon (Insert/remove Breakpoint Button)** is used to toggles the breakpoint on the current line.
* **Disable/Enable Breakpoints Icon** (Enable/Disable Breakpoint Button) is used to enables/disables the breakpoint on the current line.
* **Disable all Breakpoints Icon ( Disable All Breakpoints Button )** is used todisables all breakpoints in the program.
* **Kill all Breakpoints Icon (Kill All Breakpoints Button )** is used to removes all breakpoints in the program.

****

Step over Icon

****

**Disable Breakpoints**

**Disable all Breakpoints**

***Figure 16: Various Icons in Keil IDE***

**Various Windows in Keil µVision IDE:**

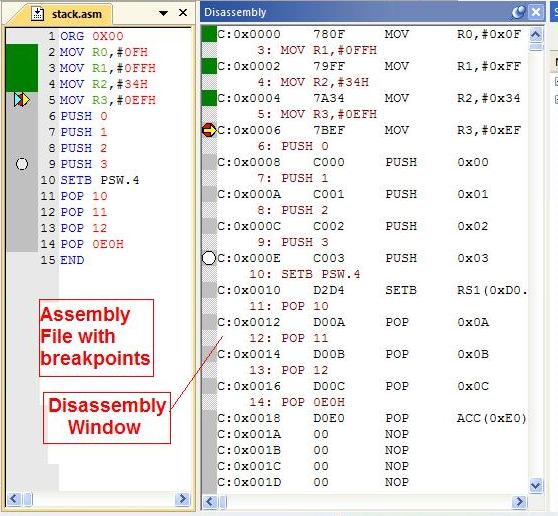
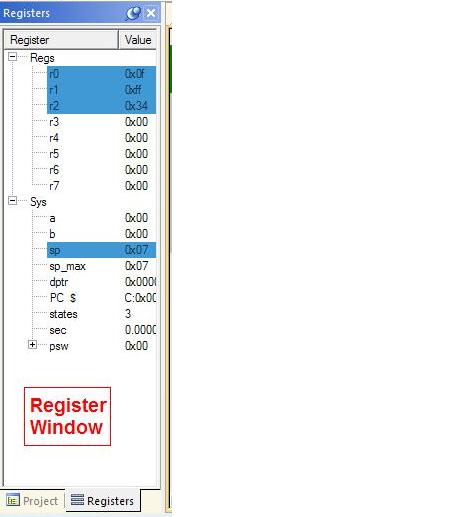
The various windows available in µVision IDE are explained in detail in this section. µVision4 enables the user to open multiple versions of windows such as memory and variable watch windows. These windows can be docked together or 'floated' anywhere on the workspace.

* **Register Window**

When the debugger is running, the Register Window is displayed in the left pane. The Register Tab of the Project Workspace Window displays the microcontroller registers and their contents. Every change in the register bank values, A and B registers, Stack Pointer, DPTR, Program Counter, Program Status Word and the number of states taken to execute the program are displayed here. All registers and flags are updated as each instruction executes. Results display in the Register Tab of the Project Workspace. As the control is stepped through the program, affected registers are highlighted.

* **Disassembly Window**

The Disassembly Window shows mixed high-level source code and its associated assembler code. The disassembly window displays the address of the memory location, instructions and their corresponding opcodes. The hex values in the disassembly window and the memory window remain the same



***Figure 17: Register and Disassembly windows***

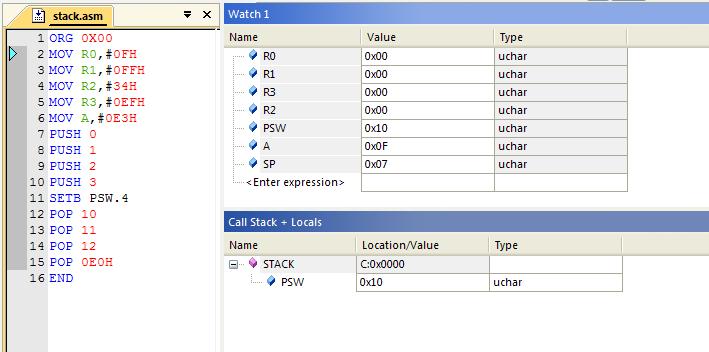
* **Watch and Call Stack Windows**

The µVision debugger provides a Watch and Call Stack Window that shows:

* The values of local variables,
* The values of watch expressions (in two windows),
* The call stack for the program.

The Watch window allows the user to evaluate symbols, registers and expressions. The window displays the item name, value, and type.

This window can be opened by clicking on the Watch Windows Icon in the toolbar or by selecting the Watch Windows option in the View menu. The Call Stack tab lists the program call tree. A function is double – clicked to show its invocation [1].

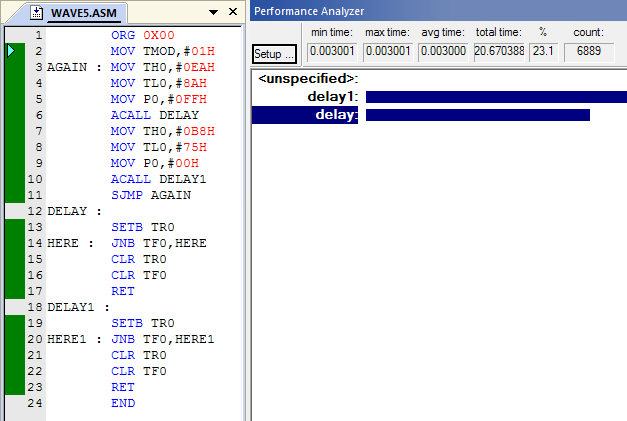


***Figure 18: Watch and Call stack Windows***

* **Performance Analyzer Window**

The performance analyzer records and displays execution times for functions and program blocks that are selected.

Bar graphs display the time spent in each part of the program. The following array of operations is performed to view the Performance Analyzer Window. On account of clicking the Setup Tab in the Performance Analyzer Window, Setup Performance Analyzer Window opens. Timer delay is 3ms

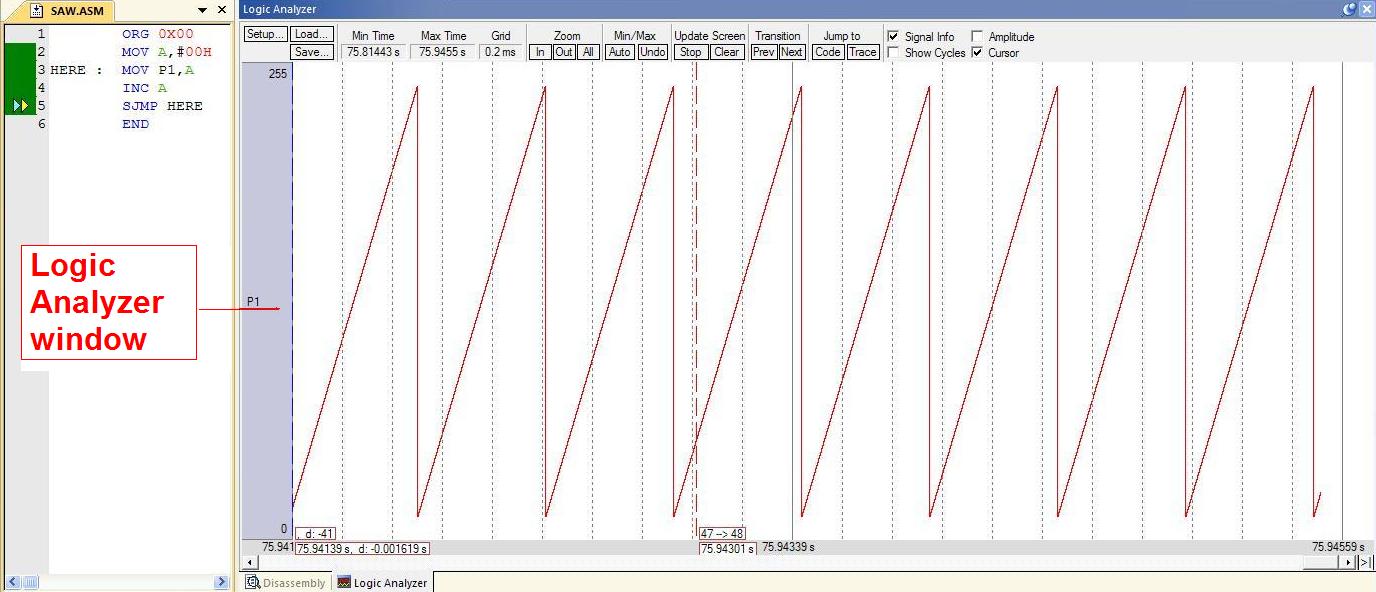


Performance Analyzer

***Figure 19: Performance Analyzer Window***

* **Logic Analyzer Window**

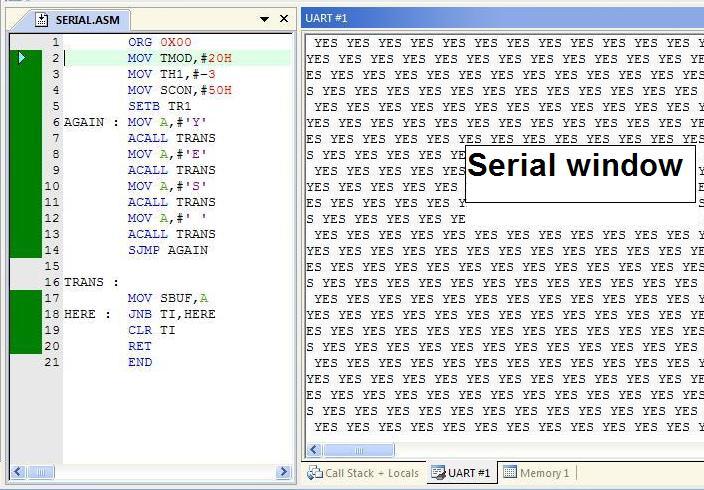
The Logic Analyzer graphically displays signals and program variables as they change over time. The logic analyzer can even take the user to the instruction or statement that is executed when the signal is changed. The following set of operations is performed to view the Logic Analyzer Window.In the event of clicking the Setup Tab in the Logic Analyzer Window, the Setup Logic Analyzer Window pops up.



***Figure 20: Logic Analyzer Window***

* **Serial Window**

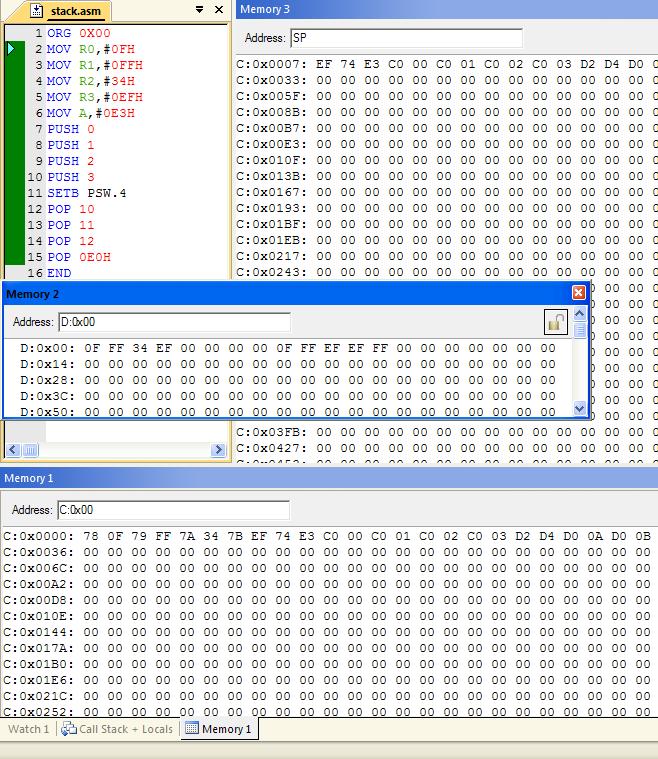
µVision provides a Serial Window that acts as an ASCII terminal for each on-chip UART. Characters that are type in the Serial Window are "received" on the simulated microcontroller. Characters that are transmitted appear in the window. μVision provides three Serial Windows, named «UART #{1|2|3}», for each simulated on-chip UART. The serial output can be assigned to a PC COM port using the ASSIGN Debugger command.



***Figure 21: Serial Window***

* **Memory Window**

The Memory Windows Icon is clicked so as to view the hex values of the instructions typed in the coding. When the address of the origin is typed in the text box which appears in the Memory Window, the hex value corresponding to the every address value can be viewed. As the program runs, µVision updates the contents of the Memory Window.



Memory windows

***Figure 22: Memory Window***

* **Breakpoints**

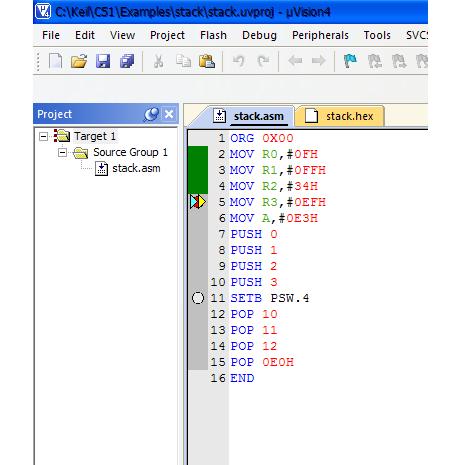
On target systems, the running application cannot be stopped until it reaches a breakpoint. Breakpoints are trigger points in the program that halt execution or execute a debugger function.

Breakpoints may be triggered by reading an instruction from program memory (an execution breakpoint), reading or writing a memory location (a memory access breakpoint), or by calculating a true value for a conditional expression (a conditional breakpoint).

The µVision IDE allows the user to set breakpoints while writing the source code and while debugging the program. Breakpoints that are set while editing are activated in the debug session. The buttons on the toolbar can be used to set breakpoints on source lines.

The breakpoints can be inserted by selecting the line of instruction where it is to be inserted and Insert Breakpoint option from the Debug menu can be chosen. The inserted breakpoint can be removed, killed or disabled.

By default, breakpoints which are created are saved and restored in subsequent debugging sessions.



***Figure 23: Breakpoints in the program***

* **Instruction Trace Window**

The Instruction Trace Window shows the history of executed instructions. The window is:

Enabled for all non-Cortex-M processor-based devices when using the µVision Simulator. Enabled only for ARM devices when debugging on target hardware.

For non-ARM devices, use the [Disassembly Window](mk:@MSITStore:C:\Keil\C51\HLP\uv4.chm::/uv4_db_dbg_disasmwin.htm) with similar features when debugging on hardware.

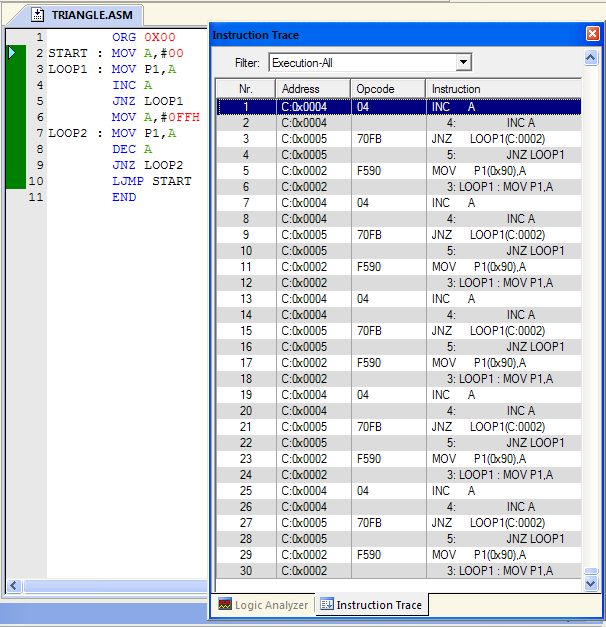
**Filter-**Allows selecting the following display modes: Execution-Mixed shows the source code and the assembler code. Execution-Asm shows the assembler code only. Execution-Source shows the source code only.

**Nr.-**Is a number reflecting the order of the executed instructions.

**Address-**Is the memory address of the instructions.

**Opcode-**Is the operation code.

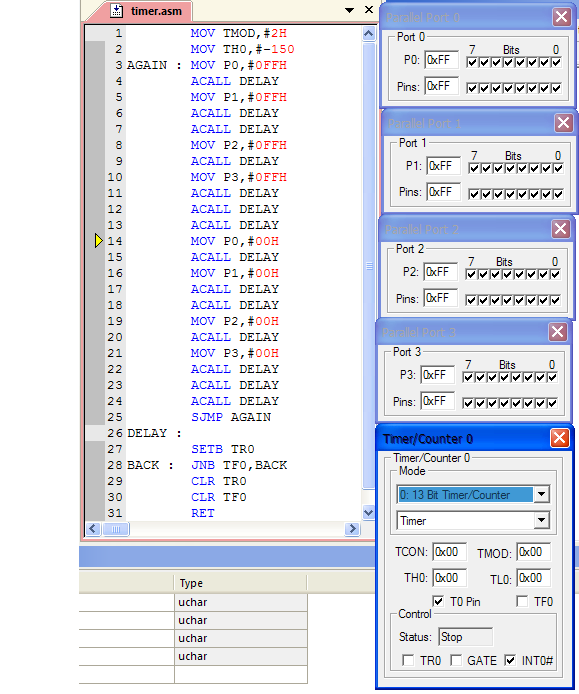
**Instructions-**Is the instructions itself. Can be the assembly or source code.



***Figure 24: Instruction trace window***

**Peripherals in Keil IDE:**

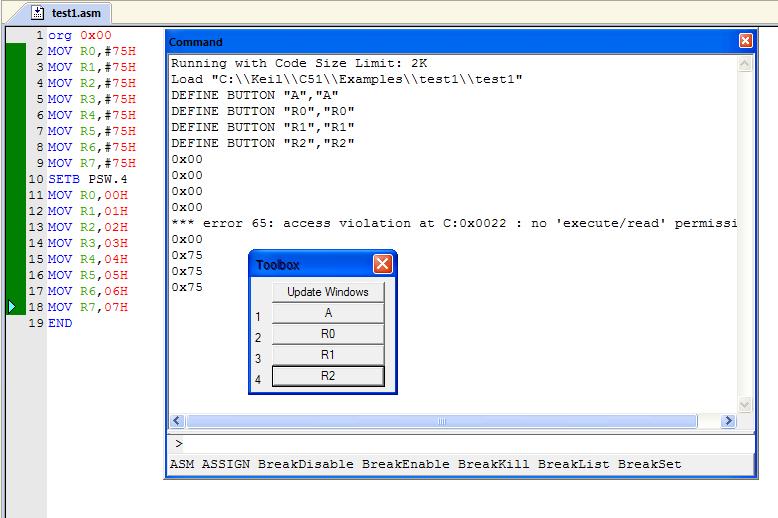
Peripherals like Timer in Keil is used to view all the information regarding Timer control register (TCON), Timer mode register (TMOD). Input / Output Ports in this Keil are used to view bit details of each port during the run time of the controller. Serial peripheral is used to view the serial data transmission and reception.



***Figure 25: Peripherals like Timer and Ports***

**Toolbox in KEIL:**

The **Toolbox** contains user-configurable buttons. Click on a **Toolbox** button to execute associated [debug commands](mk:@MSITStore:C:\Keil\UV4\UV4.chm::/uv4_debug_commands.htm) or [debug functions](mk:@MSITStore:C:\Keil\UV4\UV4.chm::/uv4_debug_functions.htm). Toolbox buttons can be clicked at any time, even while running the test program. Define Toolbox buttons in the [Command Window](mk:@MSITStore:C:\Keil\UV4\UV4.chm::/uv4_db_dbg_outputwin.htm) with the DEFINE BUTTON command.

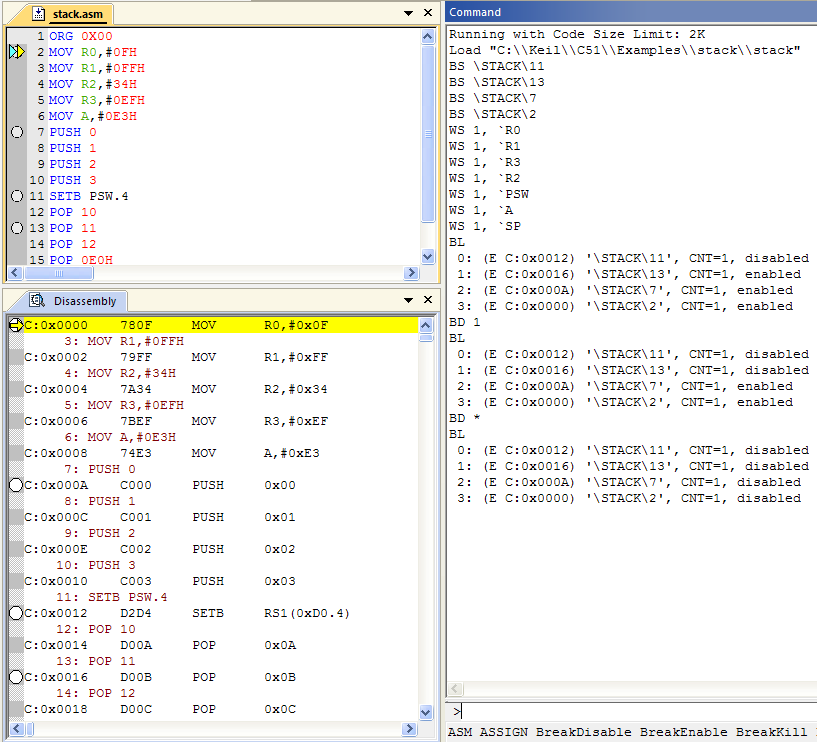


***Figure 26: Toolbox window***

**Command window:**

Command window shows generic debug output information and allows entering [debug and trace commands](mk:@MSITStore:C:\Keil\C51\HLP\uv4.chm::/uv4_debug_commands.htm) and functions to interact with the Debugger. Debug Commands can be used in [debug functions](ms-its:C:\Keil\C51\HLP\uv4.chm::/uv4_debug_functions.htm) and in the [Command Window](ms-its:C:\Keil\C51\HLP\uv4.chm::/uv4_db_dbg_outputwin.htm). The syntax generator displays matching commands, options, and parameters when using the Command Window. The list of likely commands is reduced to coincide with the typed character.

* [**BreakAccess**](http://www.keil.com/support/man/docs/uv4/uv4_cm_breakaccess.htm)**(BA**)- Adds an access breakpoint with a *length* parameter to the breakpoint list.
* [**BreakDisable**](http://www.keil.com/support/man/docs/uv4/uv4_cm_breakdisable.htm) **(BD)-**Disables one or **more** breakpoints.
* [**BreakEnable**](http://www.keil.com/support/man/docs/uv4/uv4_cm_breakenable.htm) **(BE)-**Enables one or more breakpoints.
* [**BreakKill**](http://www.keil.com/support/man/docs/uv4/uv4_cm_breakkill.htm) **(BK)-**Removes one or more breakpoints from the breakpoint list.
* [**BreakList**](http://www.keil.com/support/man/docs/uv4/uv4_cm_breaklist.htm) **(BL)-**Lists the current breakpoints.
* [**BreakSet**](http://www.keil.com/support/man/docs/uv4/uv4_cm_breakset.htm) **-**Adds a breakpoint **expression** to the list of breakpoints.

****

***Figure 27: Breakpoint usage through Command window***

**Result:**

The µVision Debugger from Keil supports simulation using only PC or laptop, and debugging using the target system and a debugger interface. µVision includes traditional features as well as sophisticated features like watch windows and logic analyzer and other windows. Thus, an insight of µVision IDE from Keil, a very effective tool to create embedded applications, has been gained.

**ASSEMBLY LANGUAGE PROGRAMMING**

**WITH KEIL**

**EXP NO.2**

**DATE:17/10/14**

**AIM**:

To develop assembly language code for 8051 for different modes of timer/counter operation, serial communication, Waveform generation with ports interrupts and to simulate it using Keil µVision IDE.

**TIMERS**:

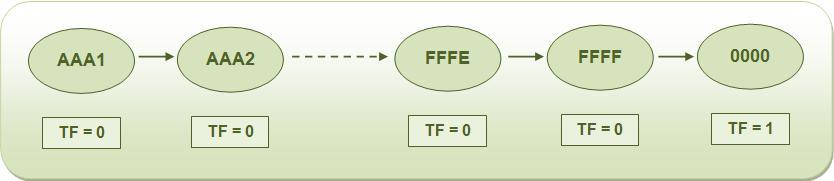
Timer is a clock that controls the sequence of an event while counting in fixed intervals of time. A Timer is used for producing precise time delay. Secondly, it can be used to repeat or initiate an action after/at a known period of time. This feature is very commonly used in several applications.

**8051 Timers and registers:**

[AT89C51](http://www.engineersgarage.com/at89c51-or-89c51-microcontroller) microcontroller has two Timers designated as Timer0 and Timer1. Each of these timers is assigned a 16-bit register. The value of a Timer register increases by one every time a timer counts. Timer takes a time period of one machine cycle to count one. (Machine cycle is a unit that refers to the time required by the microcontroller to execute instructions.) This means that the maximum number of times a timer can count without repeating is 216, i.e.,**65536**. So the maximum allowed counts in value of Timer registers can be from 0000H to FFFFH. Since [**8051**](http://www.engineersgarage.com/8051-microcontroller)is an 8 bit controller**,**the registers of 8051 Timers are accessed as two different registers; one for lower byte and other for higher byte. For example, register of Timer0 is accessed as **TL0** for lower byte and **TH0** for higher byte. Similarly **TL1**and **TH1** are registers assigned to Timer 1.

The registers of Timers are loaded with some initial value. The value of a Timer register increases by one after every machine cycle. One machine cycle duration is the 1/12th of the frequency of the crystal attached to the controller. Similarly if an 11.0592 MHz crystal is used, operating frequency of Timer is 921.6 KHz and the time period is **1.085 µs**. If no value is loaded into the Timer, it starts counting from 0000H.

When the Timer reaches FFFFH, it reloads to 0000H. This roll over is communicated to the controller by raising a flagcorresponding to that Timer, i.e., a flag bit is raised (set high) when the timer starts counting from 0000H again. **TF0** and **TF1** are the Timer flags corresponding to Timers 0 and 1. These flags must be cleared (set low) by software every time they are raised. The Timer may terminate updating register values after a roll over or continue with its operation.



***Figure 28: Timer operation***

**Starting or stopping a Timer:**

For every Timer, there is a corresponding Timer control bit which can be set or cleared by the program to start or stop the Timer. **TR0** and **TR1** are the control bits for Timers 0 and 1 respectively. Setting the control bit would start the Timer.

*TR0 = 1;*         starts Timer 0 *TR1 = 1;*         starts Timer 1

Clearing the control bit would stop the Timer.

*TR0 = 0;*         stops Timer 0 *TR1 = 0;*         stops Timer1

**Configuring a Timer:**

A register called **TMOD (Timer Mode)** is used for configuring the Timers for the desired operation. TMOD is an 8-bit register with following bit configuration:

http://www.engineersgarage.com/sites/default/files/imagecache/Original/wysiwyg_imageupload/1/TMOD%20Register_0.jpg

***Figure 29: TMOD Register***

The lower four bits (TMOD.0 – TMOD.3) are used to configure Timer 0 while the higher four bits (TMOD.4 – TMOD.7) are for Timer 1. When GATE is high, the corresponding Timer is enabled only when there is an interrupt at corresponding INTx pin of AT89C51 controller and Timer control bit is high. Otherwise only setting Timer control bit is sufficient to start the Timer. If C/T is low (0), Timer is used for time keeping, i.e., Timer updates its value automatically corresponding to 8051 clock source. When C/T is high (1), Timer is used as counter, i.e., it updates its value when it receives pulse from outside the 8051 controller.M1 and M0 bits decide the Timer modes. There are four Timer modes designated as Modes 0, 1, 2 and 3. Modes 1 and 2 are most commonly used while working with Timers.

**Programming 8051 Timers:**

  The programming of 8051 Timers can be done by using either polling method or by using interrupt. In polling, the microcontroller keeps monitoring the status of Timer flag. While doing so, it does no other operation and consumes all its processing time in checking the Timer flag until it is raised on a rollover. In interrupt method controller responds to only when the Timer flag is raised. The interrupt method prevents the wastage of controller’s processing time unlike polling method. Polling is mostly used for time delay generation and interrupt method is more useful when waveforms are to be generated or some action has to be repeated in fixed delays.

**Polling Method:**

 The polling method involves the following algorithm:

1.      Configure the Timer mode by passing a hex value into the TMOD register. This will tell the controller about which Timer is be used; the mode of Timer; operation (to be used as timer or counter); and whether external interrupt is required to start Timer.

2.      Load the initial values in the Timer low TLx and high THx byte. (x = 0/1)

3.      Start the Timer by setting TRx bit.

4.      Wait while the Timer flag TFx is raised.

5.      Clear the Timer flag. The Timer flag is raised when Timer rolls over from FFFFH to 0000H. If the Timer is not stopped, it will start updating from 0000H in case of modes 0& 1 while with initial value in case of mode 2. If TFx is not cleared, controller will not be able to detect next rollover.

6.      Stop the Timer by clearing TRx bit. If TRx bit is not cleared the Timer will restart updating from 0000H after the rollover in case of modes 0 and 1 while with initial value in case of mode 2.

**Different modes of a Timer:**

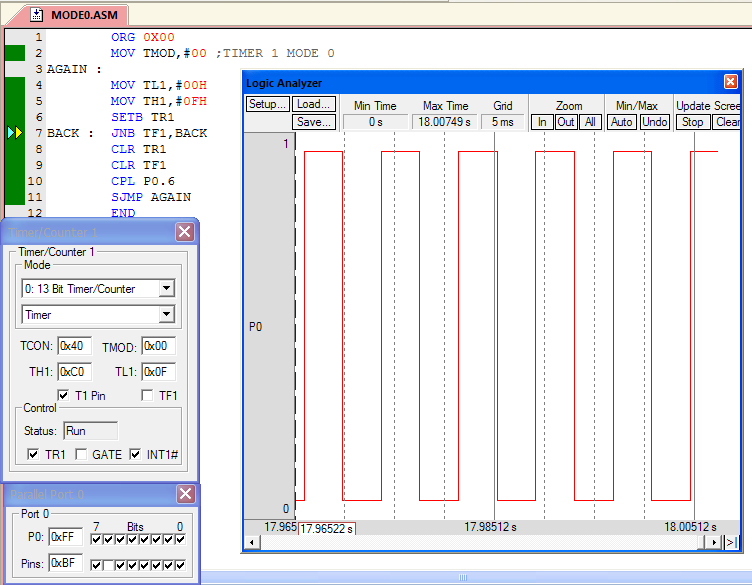
There are four Timer modes designated as Modes 0, 1, 2 and 3. A particular mode is selected by configuring the M1 & M0 bits of TMOD register.

|  |  |  |  |
| --- | --- | --- | --- |
| **Mode** | **M1** | **M0** | **Operation** |
| **Mode 0** | 0 | 0 | 13-bit Timer |
| **Mode 1** | 0 | 1 | 16-bit Timer |
| **Mode 2** | 1 | 0 | 8-bit Auto Reload |
| **Mode 3** | 1 | 1 | Split Timer Mode |

**(i) Mode 0: 13-bit Timer**

  Mode 0 is a 13 bit Timer mode and uses 8 bits of high byte and 5 bit prescaler of low byte. The value that the Timer can update in mode0 is from 0000H to 1FFFH. The 5 bits of lower byte append with the bits of higher byte. The Timer rolls over from 1FFFH to 0000H to raise the Timer flag.

Ex: Generating a square waveform with delay of 16.72ms with timer 1 operating in mode 0

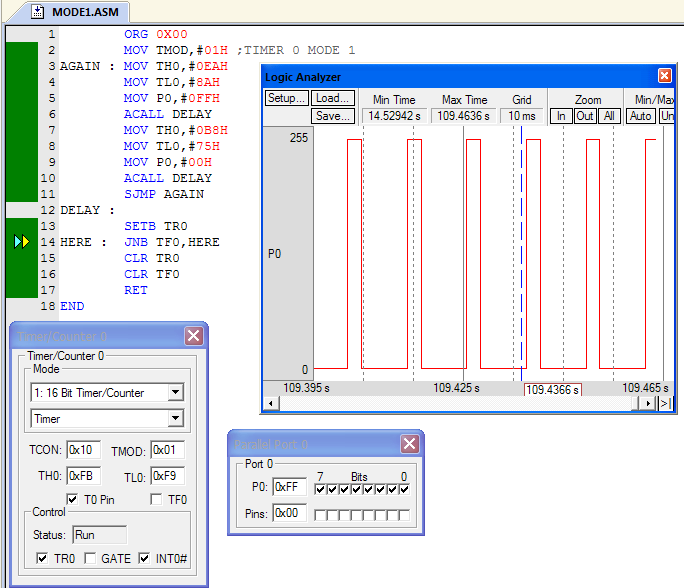


***Figure 30: Timer 1 operating in mode 0 and generating a square waveform***

**(ii)Mode 1: 16-bit Timer**

Mode1 is one of the most commonly used Timer modes. It allows all 16 bits to be used for the Timer and so it allows values to vary from 0000H to FFFFH.This mode is mostly used mode for timers for larger delays.

Ex: Generating a waveform using timer 0 mode 1 with delay of 3ms for on time and 10ms for off time.

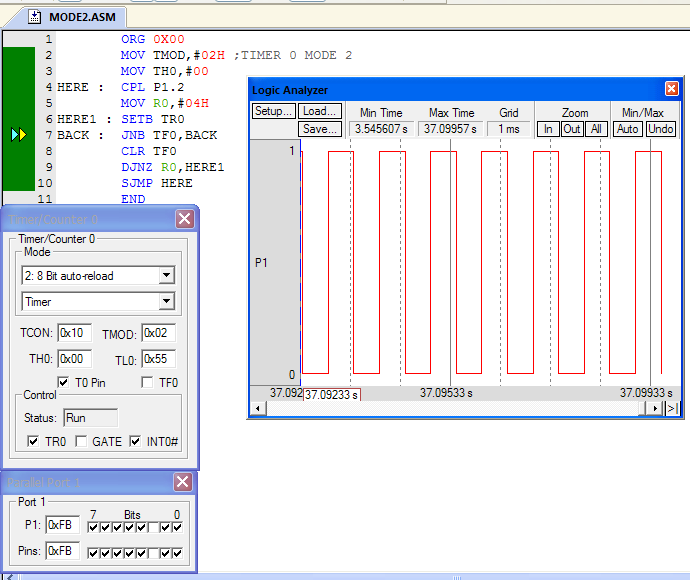


***Figure 31: Timer 0 operating in Mode 1 and generating a waveform***

**(iii) Mode 2 : 8-bit Auto Reload**

Mode 2 is an 8 bit mode. The initial value is loaded into the higher byte. A copy of the same is passed to the lower byte. The Timer can update from 00H to FFH. The Timer rolls over from FFH to initial value automatically. Mode 2 is commonly used for setting baud rates for [serial communication](http://www.engineersgarage.com/microcontroller/8051projects/interface-serialport-RS232-AT89C51).

Ex: Generating a square waveform using timer 0 operating in mode 2 (auto reload mode) with delay of 0.5ms assuming XTAL =22MHz

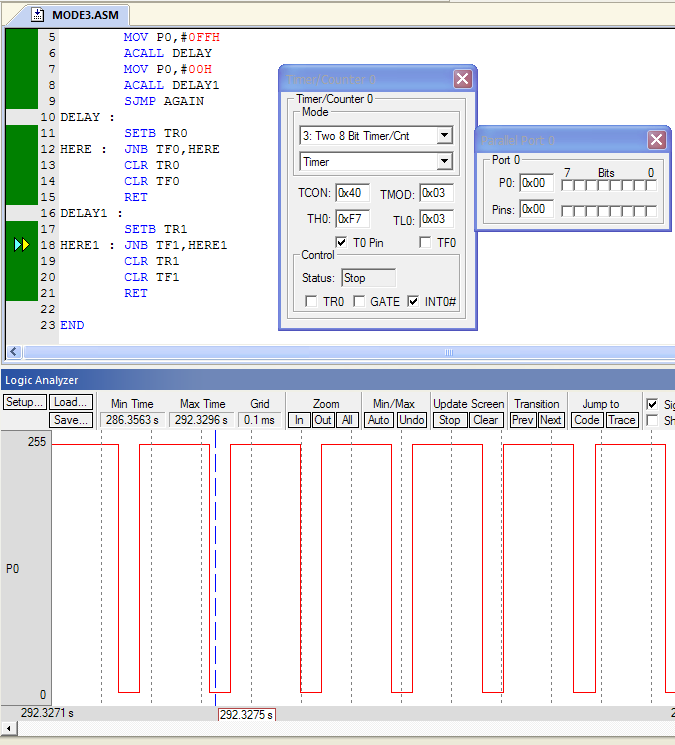


***Figure 32: Timer 0 operating in mode 2 and generating a square waveform***

**(iv) Mode 3: Split Timer**

In mode 3, also known as split mode, the Timer breaks into two 8-bit Timers that can count from 00H up to FFH. The initial values are loaded into the higher byte and lower byte of the Timer. In this case the start and flag bits associated with the other Timer are now associated with high byte Timer. So one cannot start or stop the other Timer. The other Timer can be used in modes 0, 1 or 2 and is updated automatically for every machine cycle. For example, if Timer0 is used in split mode, TH0 and TL0 will become separate Timers. The start and flag bits associated with Timer1 will now be associated with the TH0. Timer 1 cannot be stopped or started, but will be updated for every machine cycle. Split mode is useful when two Timers are required in addition to a baud rate generator.

Ex: Using Timer in split mode and generating different delays for on time with 10ms and off time with 3ms using TH0 and TL0.



***Figure 33: Timer operating in mode 3 and generating waveform***

**SERIAL COMMUNICATION:**

Serial data communication uses two methods

* Synchronous method transfers a block of data at a time
* Asynchronous method transfers a single byte at a time

SBUF is an 8-bit register used solely for serial communication

For a byte data to be transferred via the TxD line, it must be placed in the SBUF Register. The moment a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD line. SBUF holds the byte of data when it is received by 8051 RxD line. When the bits are received serially via RxD, the 8051 deframes it by eliminating the stop and start bits, making a bytes out of the data received, and then placing it in SBUF.SCON is an 8-bit register used to program the start bit, stop bit, and data bits of data framing, among other things like whether date is transmitted or received successfully.

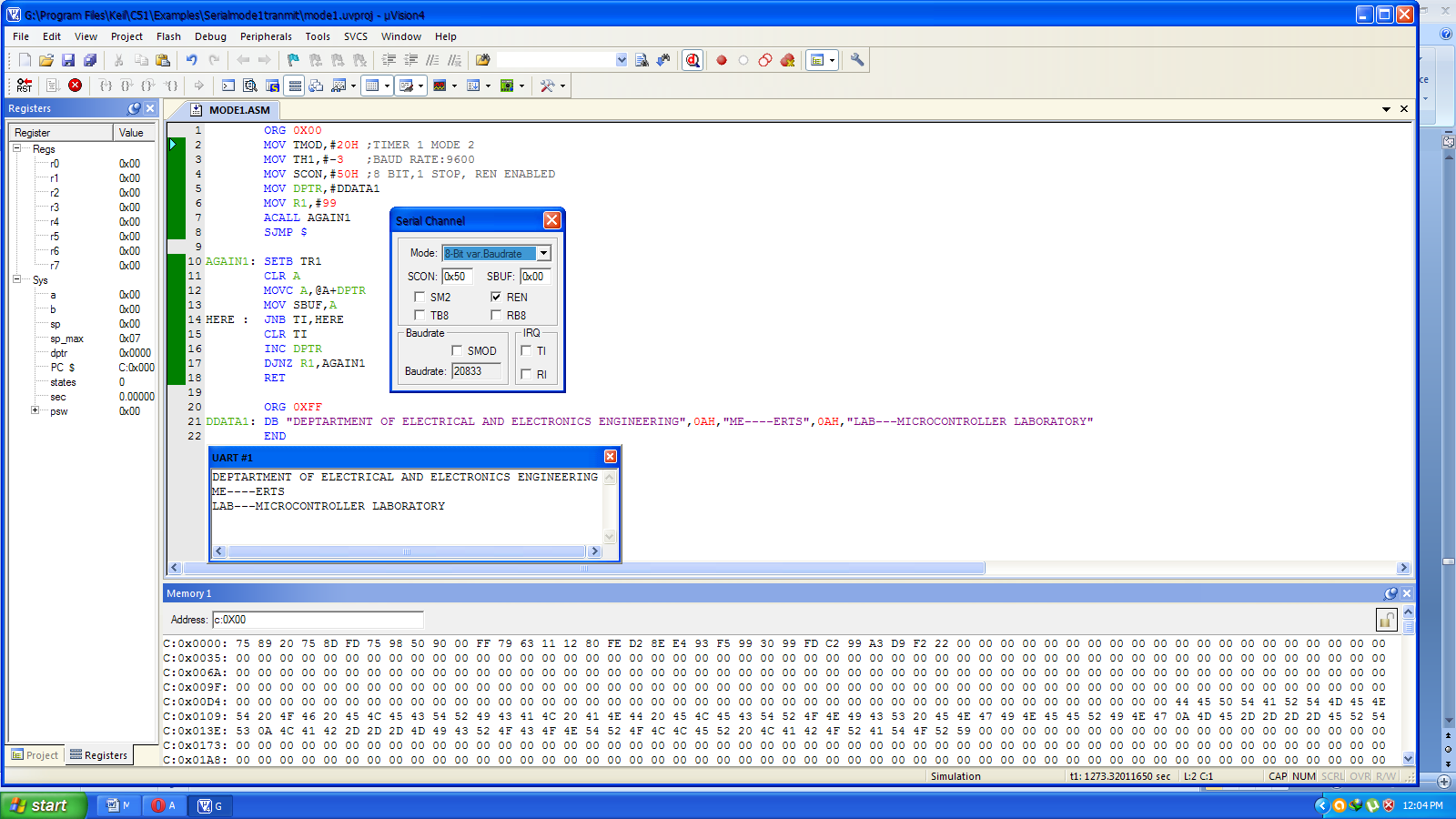
**SCON SFR and function of each bit in SFR:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit** | **Name** | **Bit Address** | **Explanation of Function** |
| 7 | SM0 | 9Fh | Serial port mode bit 0 |
| 6 | SM1 | 9Eh | Serial port mode bit 1. |
| 5 | SM2 | 9Dh | Multiprocessor Communications Enable (explained later) |
| 4 | REN | 9Ch | Receiver Enable. This bit must be set in order to receive characters. |
| 3 | TB8 | 9Bh | Transmit bit 8. The 9th bit to transmit in mode 2 and 3. |
| 2 | RB8 | 9Ah | Receive bit 8. The 9th bit received in mode 2 and 3. |
| 1 | TI | 99h | Transmit Flag. Set when a byte has been completely transmitted. |
| 0 | RI | 98h | Receive Flag. Set when a byte has been completely received. |

Additionally, it is necessary to define the function of SM0 and SM1 by an additional table:

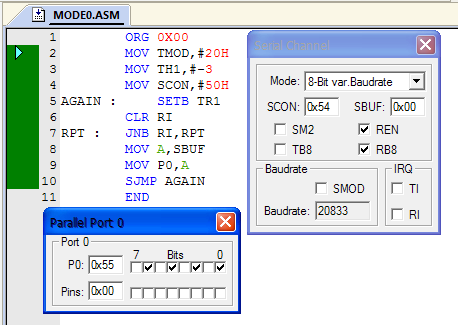
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SM0** | **SM1** | **Serial Mode** | **Explanation** | **Baud Rate** |
| 0 | 0 | 0 | 8-bit Shift Register | Oscillator / 12 |
| 0 | 1 | 1 | 8-bit UART | Set by Timer 1 (\*) |
| 1 | 0 | 2 | 9-bit UART | Oscillator / 64 (\*) |
| 1 | 1 | 3 | 9-bit UART | Set by Timer 1 (\*) |

(\*) Note: The baud rate indicated in this table is doubled if PCON.7 (SMOD) is set.

Ex: Serially transmitting a text to UART.

***Figure 34: Serially transmit data***

Ex: Serial receives data from UART and send it to port 0 continuously.



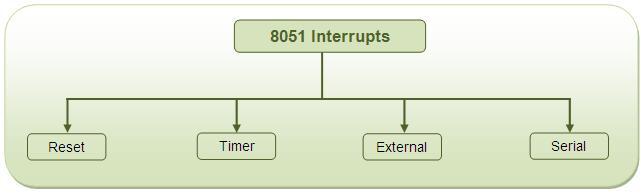
***Figure 35: Serially receive data***

**INTERRUPTS:**

The interrupts refer to a notification, communicated to the controller, by a hardware device or software, on receipt of which controller momentarily stops and responds to the interrupt. Whenever an interrupt occurs the controller completes the execution of the current instruction and starts the execution of an **Interrupt Service Routine** (ISR) or Interrupt Handler. **ISR** is a piece of code that tells the processor or controller what to do when the interrupt occurs. After the execution of ISR, controller returns back to the instruction it has jumped from (before the interrupt was received). The interrupts can be either **hardware interrupts**or**software interrupts.**

**8051 Interrupts**

The 8051 controller has six hardware interrupts of which five are available to the programmer. These are as follows:

  
***Figure 36: 8051 Interrupts***

**1. RESET interrupt** - This is also known as Power on Reset (POR). When the RESET interrupt is received, the controller restarts executing code from 0000H location. This is an interrupt which is not available to or, better to say, need not be available to the programmer.

**2. Timer interrupts** - Each Timer is associated with a Timer interrupt. A timer interrupt notifies the microcontroller that the corresponding Timer has finished counting.

**3. External interrupts** - There are two external interrupts EX0 and EX1 to serve external devices. Both these interrupts are active low. In [AT89C51](http://www.engineersgarage.com/at89c51-or-89c51-microcontroller), P3.2 (INT0) and P3.3 (INT1) pins are available for external interrupts 0 and 1 respectively. An external interrupt notifies the microcontroller that an external device needs its service.

**4. Serial interrupt** - This interrupt is used for [serial communication](http://www.engineersgarage.com/microcontroller/8051projects/interface-serialport-RS232-AT89C51). When enabled, it notifies the controller whether a byte has been received or transmitted.

**Programming Interrupts:**

While programming interrupts, first thing to do is to specify the microcontroller which interrupts must be served. This is done by configuring the Interrupt Enable (IE) register which enables or disables the various available interrupts. The Interrupt Enable register has following bits to enable/disable the hardware interrupts of the 8051 controller.

http://www.engineersgarage.com/sites/default/files/imagecache/Original/wysiwyg_imageupload/1/interrupt%20enable%20register_0.JPG

***Figure 37: Interrupt Enable (IE) register***

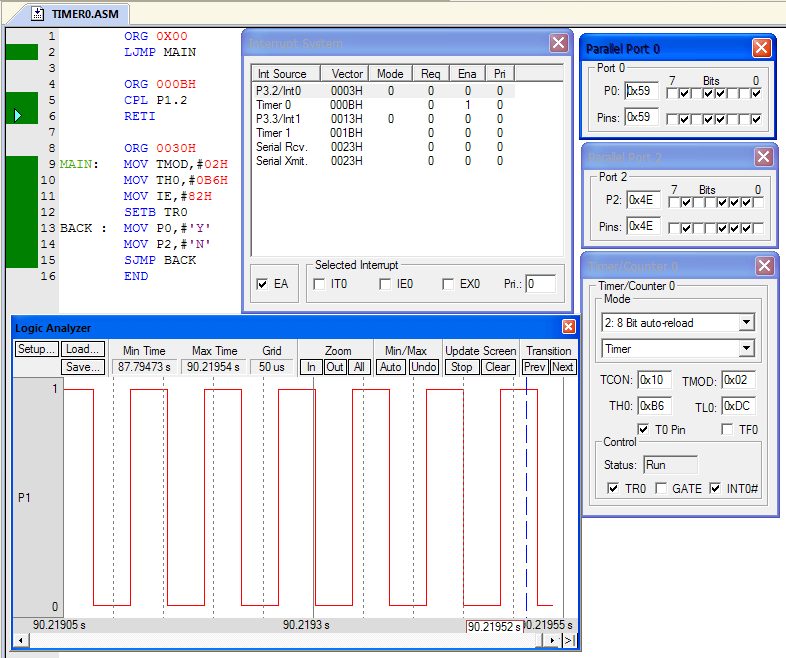
To enable any of the interrupts, first the EA bit must be set to 1. After that the bits corresponding to the desired interrupts are enabled. ET0, ET1 and ET2 bits are used to enable the Timer Interrupts 0, 1 and 2, respectively. In AT89C51, there are only two timers, so ET2 is not used. EX0 and EX1 are used to enable the external interrupts 0 and 1. ES is used for serial interrupt. EA bit acts as a lock bit. If any of the interrupt bits are enabled but EA bit is not set, the interrupt will not function. By default all the interrupts are in disabled mode. The IE register is bit addressable and individual interrupt bits can also be accessed.

**Programming Timer Interrupts**

The timer interrupts IT0 and IT1 are related to Timers 0 and 1, respectively. The interrupt programming for timers involves following steps :

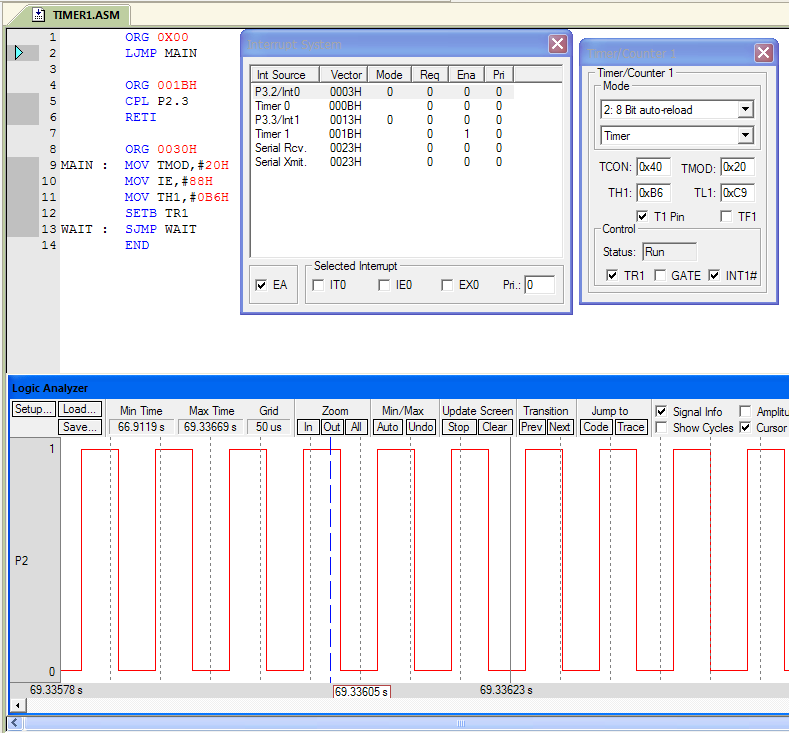
* Configure TMOD register to select timer(s) and its/their mode.
* Load initial values in THx and TLx for mode 0 and 1; or in THx only for mode 2.
* Enable Timer Interrupt by configuring bits of IE register.
* Start timer by setting timer run bit TRx.
* Write subroutine for Timer Interrupt. The interrupt number is 1 for Timer0 and 3 for Timer1.
* Note that it is not required to clear timer flag TFx.
* To stop the timer, clear TRx in the end of subroutine. Otherwise it will restart from 0000H in case of modes 0 or 1 and from initial values in case of mode 2.
* If the Timer has to run again and again, it is required to reload initial values within the routine itself (in case of mode 0 and 1). Otherwise after one cycle timer will start counting from 0000H.

Ex: Generating waveform and continuously sending data to port 0 and port 2 using timer 0 interrupt.



***Figure 38: Interrupt Timer 0 example***

Ex: Generating waveform using timer 1 interrupt.



***Figure 39: Interrupt Timer 1 example***

**Programming External Interrupts**

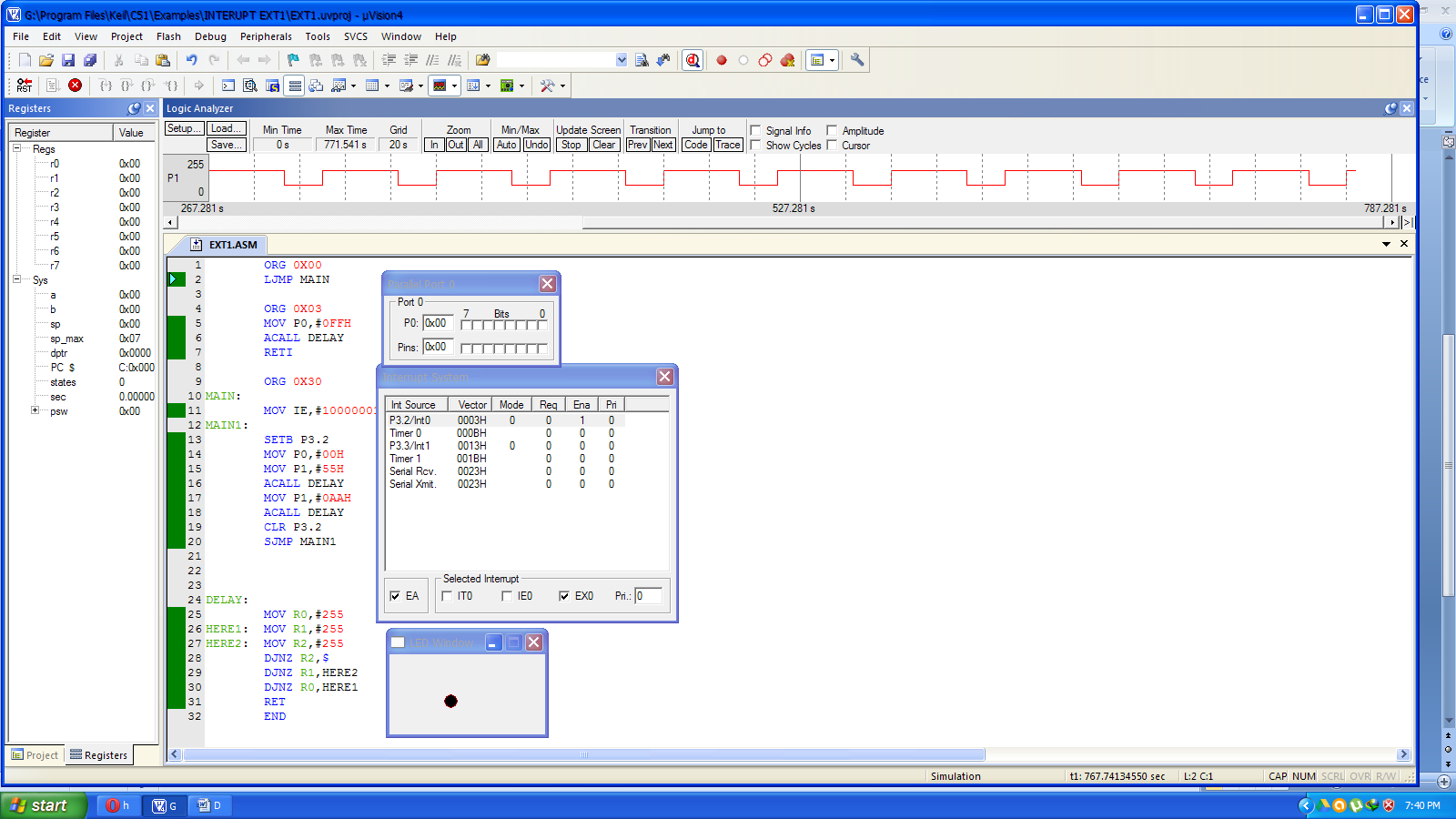
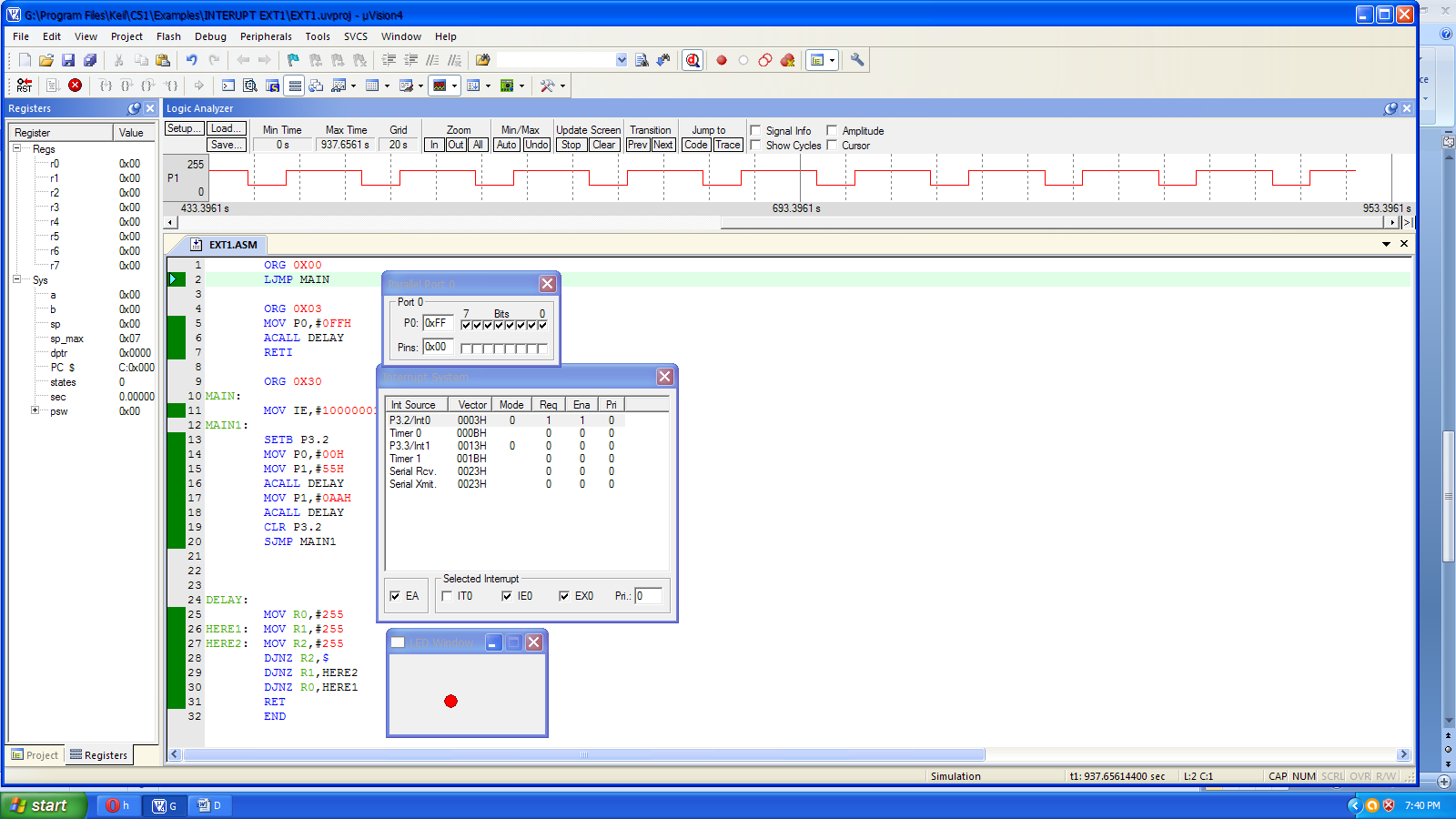
The external interrupts are the interrupts received from the (external) devices interfaced with the microcontroller. They are received at INTx pins of the controller. These can be level triggered or edge triggered. In level triggered, interrupt is enabled for a low at INTx pin; while in case of edge triggering, interrupt is enabled for a high to low transition at INTx pin. The edge or level trigger is decided by the TCON register. The TCON register has following bits:

http://www.engineersgarage.com/sites/default/files/imagecache/Original/wysiwyg_imageupload/1/TCON%20register.JPG

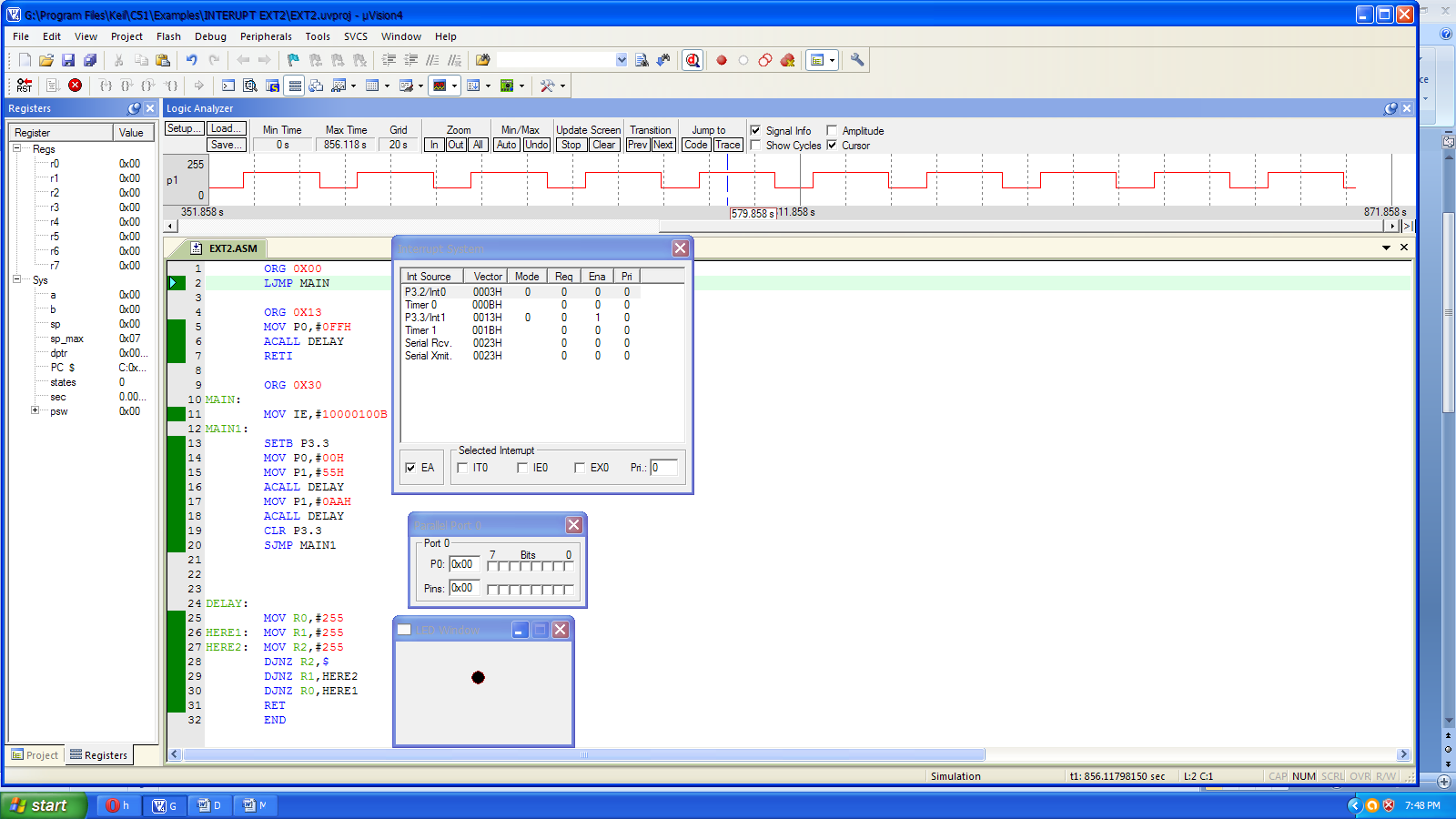
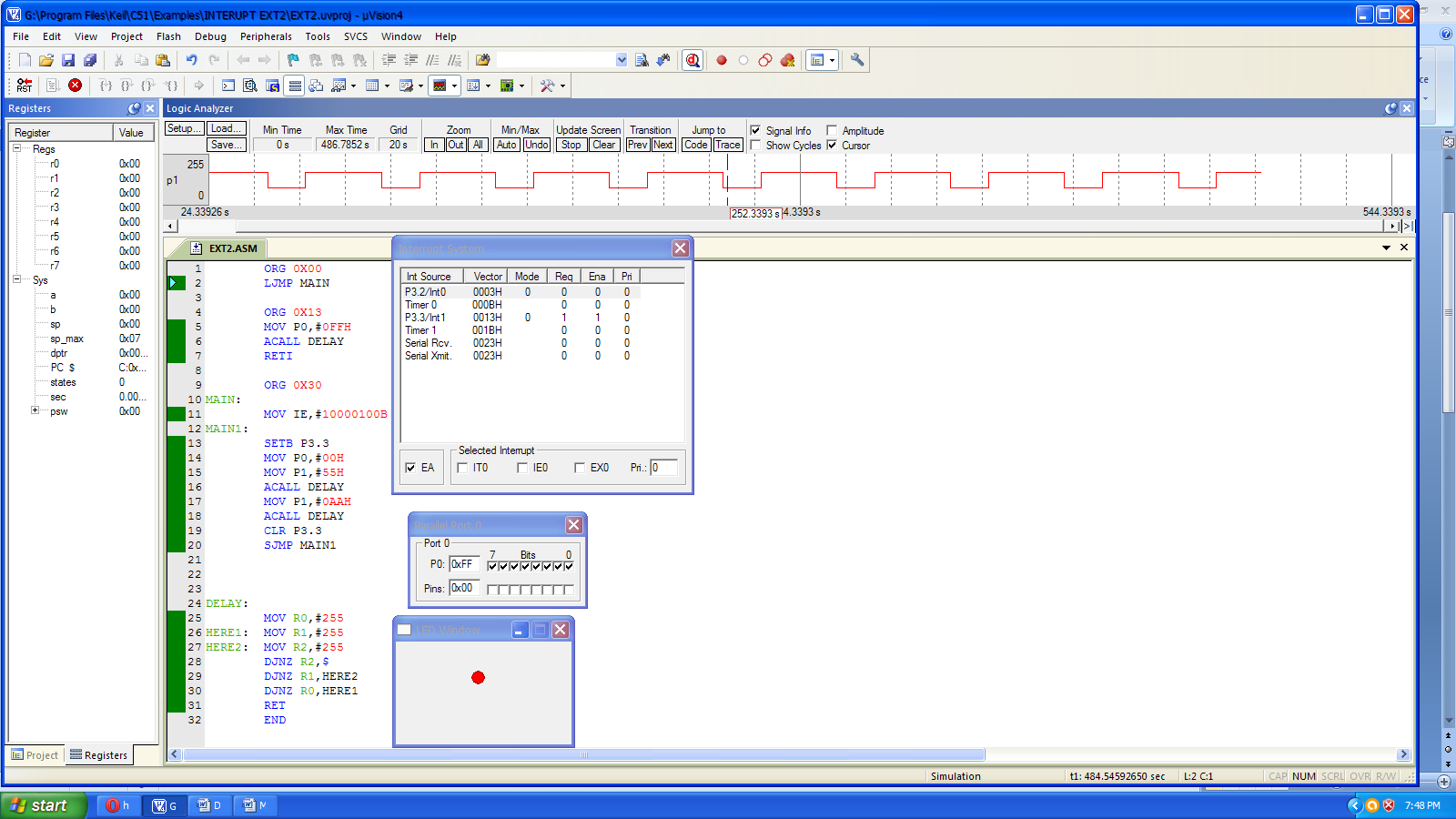
***Figure 40: TCON Register***

Setting the IT0 and IT1 bits make the external interrupt 0 and 1 edge triggered respectively. By default these bits are cleared and so external interrupt is level triggered.

Ex: Generating external interrupt 0 and all LED’s connected to port 0 glows when interrupt routine is executed

***Figure 41: External interrupt 0 example***

***Figure 42: External interrupt 1 example***

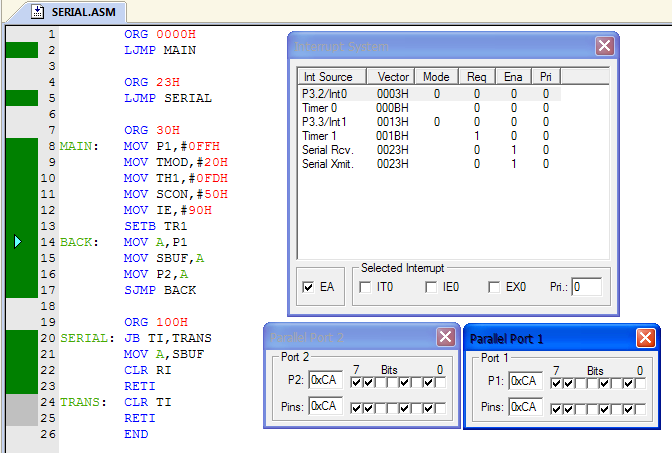
**Programming Serial Interrupt**

To use the serial interrupt the ES bit along with the EA bit is set. Whenever one byte of data is sent or received, the serial interrupt is generated and the TI or RI flag goes high. Here, the TI or RI flag needs to be cleared explicitly in the interrupt routine (written for the Serial Interrupt).

The programming of the Serial Interrupt involves the following steps:

* Enable the Serial Interrupt (configure the IE register).
* Configure SCON register.
* Write routine or function for the Serial Interrupt. The interrupt number is 4.
* Clear the RI or TI flag within the routine.

Ex: Serially transmitting data from port 1 to UART and continuously transmitting a copy of it to port 2.



***Figure 43: Interrupt Serial communication***

**Waveform generation using I/O Ports**:

All 8051 microcontrollers have 4 I/O ports each comprising 8 bits which can be configured as inputs or outputs. Accordingly, in total of 32 input/output pins enabling the microcontroller to be connected to peripheral devices are available for use.Pin configuration, i.e. whether it is to be configured as an input (1) or an output (0), depends on its logic state. In order to configure a microcontroller pin as an output, it is necessary to apply a logic zero (0) to appropriate I/O port bit. In this case, voltage level on appropriate pin will be 0.Similarly, in order to configure a microcontroller pin as an input, it is necessary to apply a logic one (1) to appropriate port. In this case, voltage level on appropriate pin will be 5V.

#### Port 0

The P0 port is characterized by two functions. If external memory is used then the lower address byte (addresses A0-A7) is applied on it. Otherwise, all bits of this port are configured as inputs/outputs.

#### Port 1

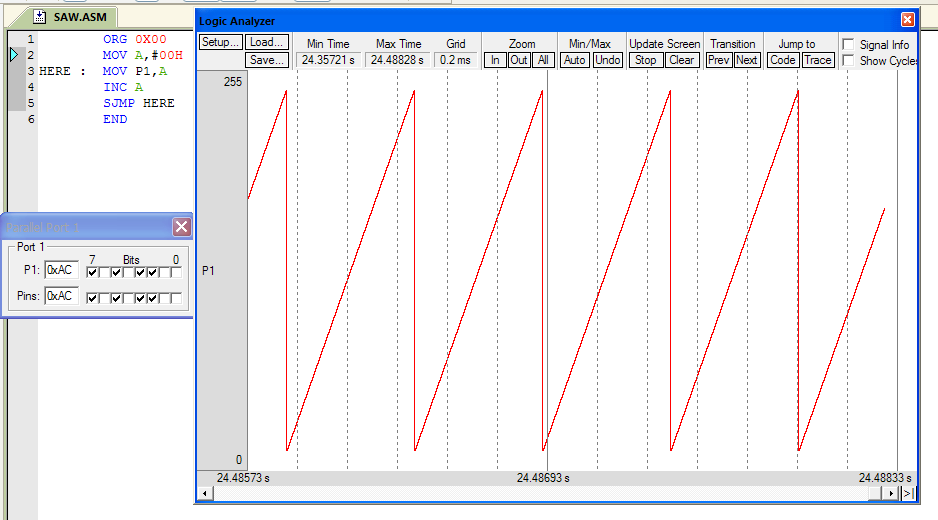
P1 is a true I/O port, because it doesn't have any alternative functions as is the case with P0, but can be cofigured as general I/O only. It has a pull-up resistor built-in and is completely compatible with TTL circuits.

#### Port 2

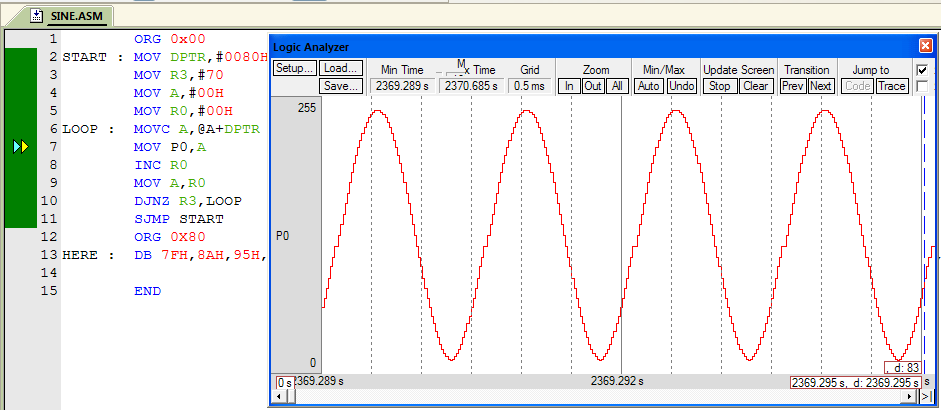
P2 acts similarly to P0 when external memory is used. Pins of this port occupy addresses intended for external memory chip. This time it is about the higher address byte with addresses A8-A15. When no memory is added, this port can be used as a general input/output port showing features similar to P1.

#### Port 3

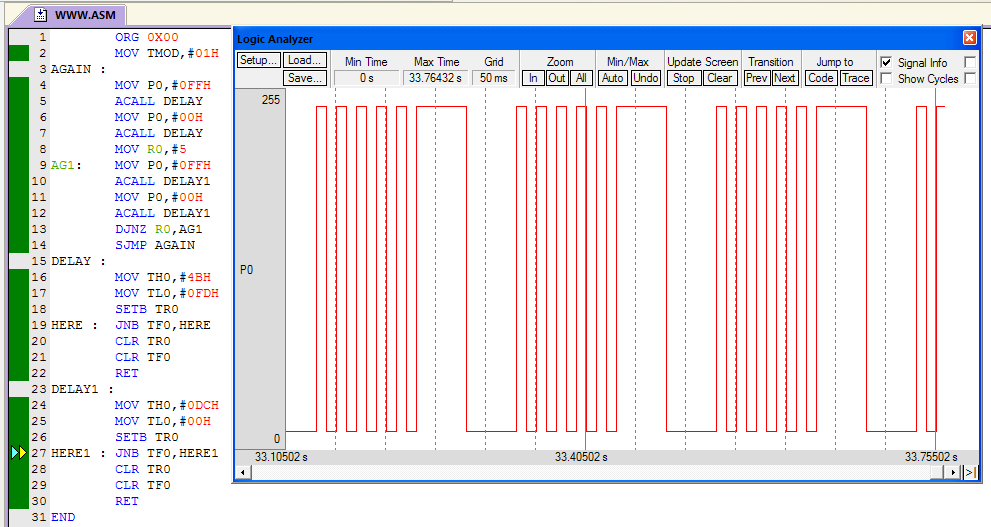
All port pins can be used as general I/O, but they also have an alternative function. In order to use these alternative functions, a logic one (1) must be applied to appropriate bit of the P3 register. In tems of hardware, this port is similar to P0, with the difference that its pins have a pull-up resistor built-in.



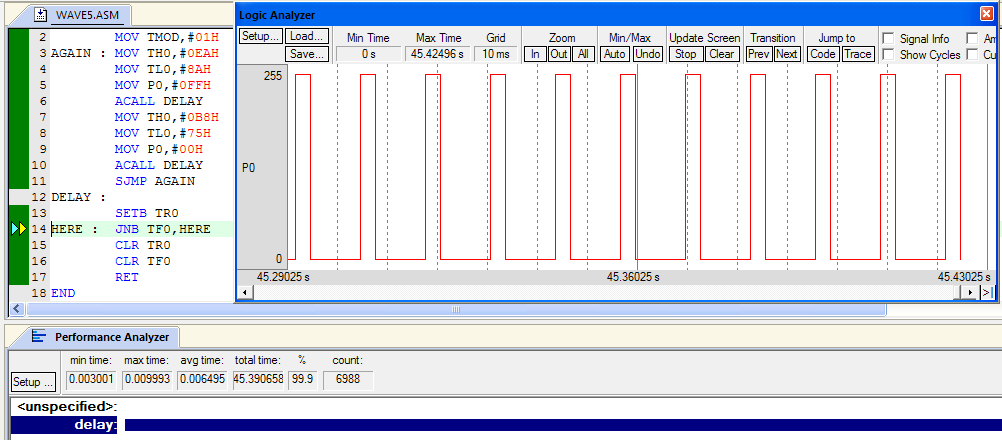
***Figure 44: Ramp waveform generation***



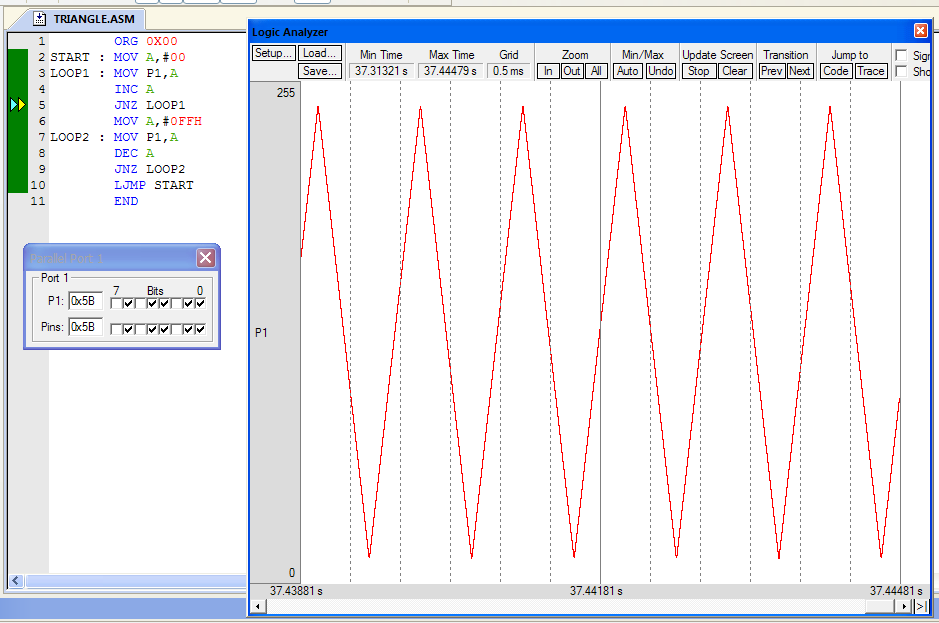
***Figure 45: Sine waveform generation***



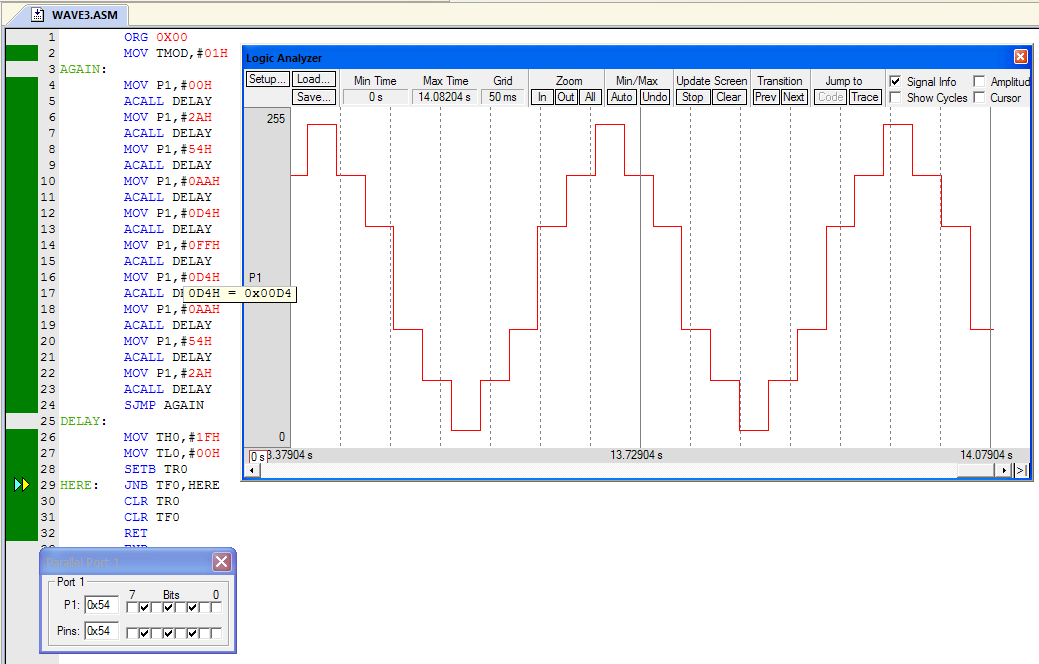
***Figure 46: Square waveform mixed with pulse waveform***



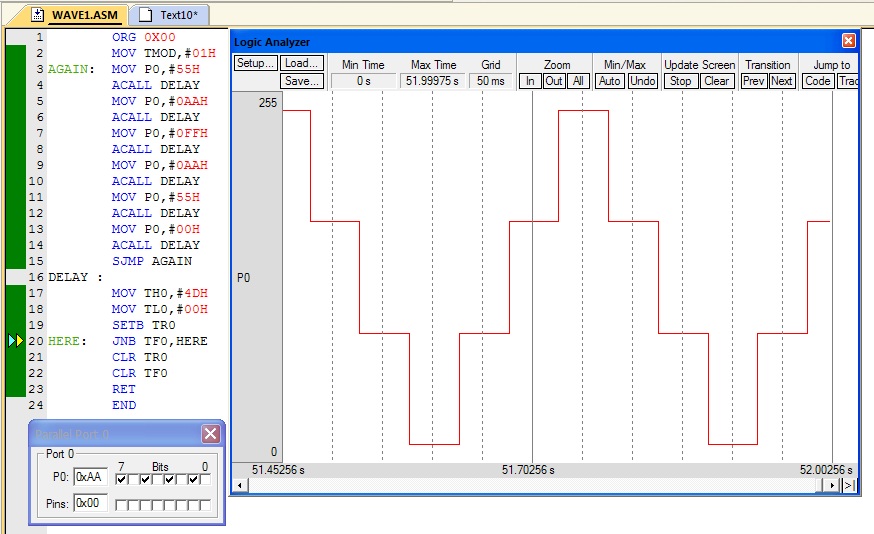
***Figure 47: Square waveform with ON time=3ms and OFF time=10ms***



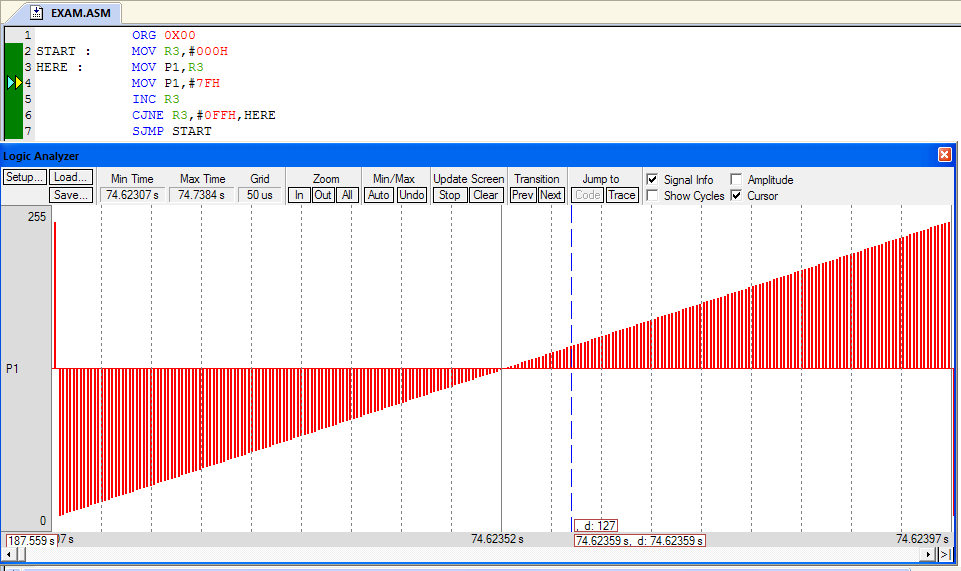
***Figure 48: Triangle waveform generation***



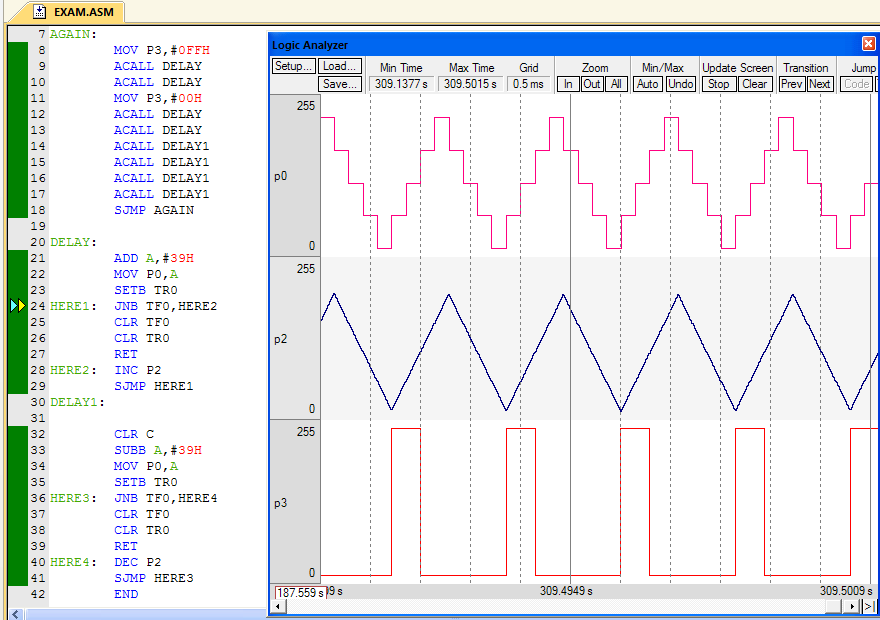
***Figure 49: Step waveform with positive and negative cycle***



***Figure 50: Step waveform with positive cycle***



***Figure 51: Waveform generation***

****

***Figure 52: Multiple Waveforms in a single program***

**Result:**

Thus the assembly language code for 8051 for different modes of timer/counter operation, serial communication, Waveform generation with ports and interrupts are done in Keil µVision IDE.